软件工程专业核心课程系列教材

软件测试实践教程

蔡建平 倪建成 高仲合 编著

清华大学出版社

软 件 工 程 专 业 核 心 课 程 系 列 教 材

软件测试实践教程

蔡建平 倪建成 高仲合 编著

清华大学出版社
北京

内 容 简 介

软件测试是软件工程的一个重要分支，它对测试人员的专业知识、专业技术、专业能力要求极高，而目前企业对测试人员的要求是要有较丰富的测试经验及较强的测试工具应用能力。本书作为《软件测试方法与技术》配套的实验教材，通过覆盖软件评测的各个环节和知识点，以主流的开源软件测试工具应用为基础，以实战能力培养为目的，为高等院校不同学历教育的软件工程专业和计算机相关专业开设软件测试课程提供了全方位的，并且是可行或可用的实践教学方案和实践教学平台以及配套的实践教学案例。

全书共12章，分为管理、静态分析、单元测试、GUI测试、性能测试及软件综合评测共6大部分。主要内容包括软件缺陷管理、软件测试管理、程序理解、代码静态分析、xUnit单元测试框架、单元覆盖测试、Java GUI基础类库应用测试、Web页面测试、Gtk+用户界面测试、单元性能测试、Web应用性能测试以及软件综合评测工具等。

掌握软件测试技术、构建软件测试环境、编写软件测试用例、开展软件测试工作并有效进行软件测试管理，无论是对于软件管理人员、开发人员、质量保证人员还是测试人员，都具有较强的现实意义。本书针对软件测试的实验内容全面，实验方案完整，实践环境建设可行，实验步骤及过程讲解清晰，实验案例丰富实用，可作为高等院校不同学历教育的软件工程及计算机相关专业的“软件测试实验课程”教材(如本科生、研究生，甚至高职生或高专生等)，也可作为软件测试实战培训教材，同时本书也是软件开发或管理人员、测试或质量保证人员非常好的自学参考书。

图书在版编目（CIP）数据

软件测试实践教程/蔡建平，倪建成，高仲合编著. --北京：清华大学出版社，2014
软件工程专业核心课程系列教材
ISBN 978-7-302-36040-7

Ⅰ. ①软…　Ⅱ. ①蔡…　②倪…　③高…　Ⅲ. ①软件–测试–教材　Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字（2014）第065978号

责任编辑：魏江江　薛　阳
封面设计：
责任校对：时翠兰
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn,010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**25.25　　**插页：**2　　**字　　数：**599千字
版　　次：2014年11月第1版　　**印　　次：**2014年11月第1次印刷
印　　数：1～0000
定　　价：0.00元

产品编号：054100-01

前　言

本书是本人编著的全国工程硕士专业学位教育指导委员会推荐教材及软件工程专业核心课程系列教材——《软件测试方法与技术》配套的实践教材。

《软件测试实验指导教程》自 2009 年 11 月发行到 2013 年 2 月销售殆尽，有二十多所高等院校将它指定为实验教材或实验参考书籍。在这期间，作者一直密切地与高校教材的使用者——授课老师交流，听取他们对本教材的意见。同时，也密切关注企业应用开源软件测试工具进行软件测试的情况，了解企业常用的开源测试工具都有哪些，并基于此做了部分调整和补充。最后，为保持其延续性，本书在教材结构和内容组织上没有做太大的调整，主要是在软件测试工具的升级、软件测试类型的覆盖、软件测试案例的补充以及软件测试工具的选型上进行了修改、补充和完善，使本教材在内容上更加新颖和完整。

软件测试是一门对工程实践要求极高，对学生动手能力要求极强的软件工程核心课程。目前许多高校不同学历教育的计算机类专业均开设了这门课程，并配套有大量学时的实验课程或额外配套的课程设计实践课程。

本书充分考虑到现代软件测试贯穿软件工程整个软件生命周期，需要用到多种测试技术、方法的实际要求，以及国内大多数院校办学条件不足，实验教学经费有限，无法全方位引进商用软件测试工具，无法开展软件测试实验室建设的实际情况，对国内外主流的开源软件测试工具进行全面分析、研究和优选，并经过十多轮实践教学的检验，来设计本书的实验教学重点和实践能力要求。

本书与国内常见的软件测试实践教材重点讲授一般商用软件测试工具的方法不同，它涉及的实验内容非常广，软件测试知识非常多，开源软件测试工具实验非常全面，并且易于剪裁或扩充，无论是对于学生工具学习，还是教师实验指导，以及培训机构实战训练都是不可多得的实验教材。

本书的出版首先要感谢清华大学出版社的大力支持和帮助。

另外，本书的完成得益于许多学生的积极参与。在这里我要特别感谢我指导的研究生乔丽平、路翠、邱聃、刘泽伟等同学，他们在资料收集，章节起草，内容组织、实验完成以及图表制作等方面做了很多工作。我还要感谢北京工业大学软件工程专业 03 至 06 级的一些学生，如 03 级的安文怡、李征和刘欣宇等，04 级的孙建和刘茜等，05 级的杨天放、时永欣、赵京超和周丰等，06 级的黄飞和霍晓珍等，本书的很多内容是来自于他们完成的软件测试课程设计实践总结报告。

其次，本书的大量内容是取材于互联网，并进行组织和修改的结果。遗憾的是很多网上资料由于转载或引用找不到原创处，在参考文献中无法准确标注，在这里一并表示感谢。

最后，对家人的感谢是必需的，教材、专著的编写离不开她们多年的支持和照顾。

目前国内软件测试的实验书籍或教材也逐渐多了起来，都有自己的特点或特色。但愿本实验教材能够一如既往地受到学生、教师以及软件工程师等广大读者的欢迎。当然，由于自身能力和水平有限，书中难免有许多不周到、不准确、疏漏或不足之处，恳请读者提出批评和建议，以便及时修正。

蔡建平

2014 年 10 月　于北京

作 者 简 介

蔡建平，原北京工业大学教授。曾在总装备部某研究所工作十余年，一直从事该所负责的全军军用共性软件系统项目的论证和研究，并在其中发挥重要作用，得到有关领导和专家的认可和好评。获军队科技进步一等奖、二等奖、三等奖多项；发表各类学术论文三十多篇，与他人合作著书一部。

在北京奥吉通科技有限公司任技术总监期间，除负责国防有关单位的软件工程、软件质量保证、软件测试以及嵌入式开发的技术咨询、提供解决方案和技术支持外，还主持开发了科锐时系列软件测试工具，并成功用于国防软件的测试。

2005 年 9 月在北京工业大学软件学院任职教授以来，除了多年讲授软件测试课程和多次对外开展嵌入式软件测试技术培训外，在学院的学科建设、专业建设，如实验室建设、实践教学改革与创新、数字艺术方向和数字媒体技术专业建设、“211 工程”建设、教育部和北京市特色专业建设，以及科研基地——科技创新平台建设等方面做了大量的工作，取得了突破性成果。

2013 年 10 月在航天中认软件测评科技（北京）有限责任公司担任专家咨询委员会副主任，负责公司的业务规划。

2014 年 1 月，在惠普国际软件人才及产业基地担任学术总监。同年 4 月，被聘为曲阜师范大学的兼职教授。

目　录

第III部分　单元测试篇

第Ⅳ部分　图形用户界面测试篇

第Ⅴ部分　性能测试篇

第Ⅵ部分　软件综合评测篇

第Ⅰ部分　管　理　篇

软件工程除了技术外，最重要的思想之一就是管理。软件测试作为软件工程的一个重要分支，其目标是保证软件生命周期中每个阶段的活动结果是正确的，这就是现代软件测试思想——全生命周期软件测试思想。软件测试管理是软件测试质量的重要保证手段，它要解决的问题是如何确保软件测试技术能使软件项目在软件生命周期内得到顺利实施，并产生预期的效果。

事实上，随着技术的发展，软件系统的规模急剧增大，采用国际协作的模式，由位于世界上不同国家不同城市的多个团队联合开发软件系统，已经成为目前软件开发的主要趋势。与之相适应，测试也需要物理上分布的多个团队共同参与。由于各个团队承担不同的任务，并有着不同的项目管理模式，为保证整个系统能得到一致、有效的质量控制，测试管理至关重要。测试管理有助于系统、规范地管理各种测试资源和测试活动，以提高测试的效率和质量。按照管理对象的不同，软件测试管理大致分为软件测试团队组织管理、软件测试计划管理、软件缺陷(错误)跟踪管理以及软件测试资源管理这四大部分。

软件测试团队组织管理，通俗地讲就是测试团队应该如何组建、人员应该如何分工与管理以及绩效应该如何考核等。

软件测试计划管理，通俗地讲就是安排好测试流程。这部分内容具体涵盖软件测试策划、软件测试技术剪裁、测试进度管理、测试成本管理等几个部分。其中：软件测试策划工作主要是指在具体测试活动实施之前做好策划工作，如起草测试大纲和测试计划；软件测试技术剪裁工作主要是指测试团队应根据软件项目的具体情况剪裁出所要实施的测试技术；测试进度管理工作主要是指排出各项测试的时间进度及人员安排，如有变动则应做相应调整；测试成本管理工作主要是开列出测试活动中会涉及的资源需求。

软件测试团队组织管理和软件测试计划管理属于软件项目管理的范畴，可根据软件测试的特点和 GB/T 15532—2008 计算机软件测试规范以及 GB/T 9386—2008 计算机软件测试文档编制规范来开展相关的管理工作。

软件缺陷(错误)跟踪管理，通俗地讲就是确保发现的缺陷(错误)已经被开发团队纠正或处理过，并且没有引入新的缺陷(错误)。具体来讲，当测试团队通过各种途径发现了文档或代码中的缺陷或错误以后，并不是交一份测试报告就草草了事，而是在递交报告以后继续督促开发团队及时关闭已知缺陷或错误(当然，如有必要，应对这些缺陷、错误做严重程度排序，以便开发团队能视轻重缓急安排处理顺序)。当开发团队关闭了测试报告中的缺陷(错误)以后，测试团队还需验证开发团队在关闭过程中有没有引入新的错误。通常，这个

过程称为回归测试。回归测试如发现问题，则继续报告给开发团队，按上述流程循环，直至回归测试最终通过。

软件测试资源管理，通俗地讲就是努力建设好测试团队的测试资源库，并对测试团队成员进行技能培训以帮助他们使用好这个测试资源库。软件测试资源库所包括的内容是测试团队在长期实践过程中逐步积累起来的经验教训、测试技巧、测试工具、规格文档以及一些经过少量修改便能推广至通用的测试脚本程序。测试资源管理工作做得越好，测试团队在实际测试过程中就越能少走弯路，测试团队内部的知识交流和传递就越充分，测试脚本或规格文档的重复开发工作也就越能被有效地避免。软件测试资源管理工作包括两部分：一个是建设，另一个是培训。建设工作大抵是收集各类测试文档、测试工具、测试脚本，也包括收集整理测试人员的会议发言、总结报告、技术心得等。培训工作大抵是通过技术讲座、正式或非正式团队会议、印发学习资料等形式进行。

软件测试管理工具是软件测试管理最重要的保证手段，对于大型系统测试来说，测试管理工具可以帮助组织测试资产、监督项目状态、集成自动化测试工具以及度量测试效果，能够为所有这些参与者提供一个交流和协作的平台，是项目管理中必不可少的。近年来，测试工具的应用越来越广泛，很多工具都能提供一定程度的测试管理功能。当然，商用软件测试管理工具不论从能力还是从成熟度来说都是有较强竞争力的，在条件许可的情况下应该作为首选。但是，开源软件测试管理工具发展越来越迅速，功能越来越强大，应用越来越广泛，是进行软件测试管理实践教学的最佳解决方案之一。本部分以开源为基础，重点讲解有关的软件测试管理工具。

第1章　软件缺陷管理

软件开发是引入软件错误或软件缺陷的过程，软件测试则是发现软件错误或软件缺陷的过程。对于大型软件来说，错误数目是非常可观的，必须借助工具才能对所发现的这些错误进行有效的管理，为软件缺陷或错误的消除或者软件质量的评价及软件开发的决策提供依据。

1.1　缺陷管理工具介绍

有些项目很简单，缺陷也就十几个、几十个，采用手工进行缺陷管理就可以了。而有些项目，特别是大型软件，会有成千上万个缺陷，甚至还有无法预计的缺陷。这时，手工进行缺陷管理就很不现实，而选用合理的缺陷管理工具便成了不可回避的问题。

本节主要介绍一些开源的缺陷管理工具，以便读者使用。例如 Mantis(免费)、Bugzilla (免费)、JIRA(免费)、TrackRecord(Compuware 公司)、ClearQuest(IBM Rational 公司)，这些都是专门的缺陷管理工具。此外，有些测试管理工具也具有缺陷管理的功能，如 HP 公司的 ALM、IBM Rational 公司的 TestManager，但这些都是商业软件。

商业软件有商业软件的好处，例如 MI 公司的 TestDirector，采用的是 B/S 构架模式，Windows 平台，可以定制流程、查询、功能域、用户角色及角色权限，可 E-mail 通知，可以生成各种报表并支持多种数据库，还可以与其他 MI 公司测试工具集成，安装配置也较为简单、有可优化的工作流、可使用 C 语言来改进优化系统。当然，开源软件也有其自身的优点：基本上都是基于 Web 的、提供源程序、免费使用、用户有更多的自由操作空间，等等。

基于 Web 的缺陷跟踪系统有很多好处，它简化了缺陷管理的相关工作。

(1) 实现地域上分散的项目人员高效协同工作，有效地降低软件测试成本，提高工作效率。

(2) 通过设置不同的用户权限，安全、准确地实现缺陷的管理和跟踪，且便于项目结束后的存档，以备将来参考。

(3) 系统维护简单，如果采用 B/S 结构，则只需要修改服务器端。现有的基于 Web 的缺陷跟踪系统所提供的功能都基本相同，但是在管理结构的实现上有一些不同，最简单的实现是一个系统有一个管理员，这个管理员负责管理所有的用户和项目。管理员的工作量

会随用户和项目的增加而增加，通信和管理成本也会增加，因此这样的系统不适合大型项目或者地域分散的软件生产商采用。

下面简单介绍几个常用的开源缺陷管理工具。

1.1.1　Bugzilla

Bugzilla(免费，跨平台)是一款开源的 Bug 追踪系统，可以用来帮助管理软件开发。Bugzilla 虽然是专门为 UNIX 定制开发的，但是在 Windows 平台下依然可以成功安装使用。

而且，Bugzilla 还能够被集成到 Testopia(测试用例管理系统)系统中。Bugzilla 的强大功能表现在以下几个方面。

(1) 强大的检索功能。

(2) 用户可配置通过 E-mail 来公布 Bug 变更。

(3) 历史变更记录。

(4) 通过跟踪和描述处理 Bug。

(5) 附件管理。

(6) 完备的产品分类方案和细致的安全策略。

(7) 安全的审核机制。

(8) 强大的后端数据库支持。

(9) Web、XML、E-mail 和控制界面。

(10) 友好的网络用户界面。

(11) 丰富多样的配置设定。

(12) 版本间向下兼容。

1.1.2　BugOnline

BugOnline(开源)是一款开源的 Bug 管理系统，功能强大，易于使用。BugOnline 基于 ASP.NET、SQL Server(包括 Express 版)及 AJAX 等技术。

BugOnline 具有如下一些特性。

(1) 在线消息及 E-mail 自动通知功能。当有新 Bug 及 Bug 分配给用户时，会自动通知用户。

(2) 优秀的人员分配、工作量统计功能。

(3) 基于项目角色的权限管理、工作规划及流程化。

(4) Bug 状态统计，便于掌控项目进度。

(5) 基于 SSL 的数据传输，确保数据不被截取，保证安全性(也可设定为非 SSL)。

(6) 强大的报表功能。

1.1.3　Bugzero

Bugzero(免费开源，跨平台)是一款多功能、基于网络并在浏览器下运行的 Bug 缺陷管理和跟踪系统，可用来记录、跟踪、并归类处理软件开发过程中出现的 Bug 和硬件系统中

存在的缺陷。 Bugzero 还是一个完整的服务管理软件，集成了服务台热线流程管理，可用来记录各种日常事务、变更请求和问题报告，并追踪和处理各种客户询问、反馈和意见。

Bugzero 提供了一个可靠的中央数据库，使得公司内部团队成员以及外部客户能在任何地点、任何时间进行协调和信息交流，并且使任何记录都有据可查。它使用户省时省力。Bugzero 不但使用方便，而且功能齐全，变通性好，能够灵活设置各种实际工作流程，满足特定业务和产品环境下的需求。这种灵活、易用的缺陷跟踪流程不仅增强了项目开发的质量，同时也提高了整个机构的生产效率。

1.1.4　其他开源缺陷管理工具

Bugtracker 是一个完整的 Bug / Issue 管理系统，以 Java Servlet 作为 Web 前台，以 MySQL 数据库作为后台。

BugFree 是借鉴微软的研发流程和 Bug 管理理念，使用 PHP+MySQL 独立编写的一个 Bug 管理系统。它简单实用、免费并且开放源代码(遵循 GNU GPL)。

JTrac 是一个开源且可高度配置的用于缺陷追踪的 Web 应用程序。它可以跟踪网络应用程序，可方便地实现定制，增加自定义字段和下拉式。其特点包括可定制的工作流程、实地许可、电子邮件集成、文件附件和详细历史记录查询。

BugNet 是一个不错的开源 Bug 跟踪和项目管理系统。

eTraxis 是基于网页的免费 Bug 跟踪系统。其主要特点是完全自定义模板、先进的过滤器、LDAP 支持、电子邮件通知、订阅报刊、提醒、灵活的权限管理、图形化的项目指标等。

1.2　缺陷管理工具 Mantis 及其应用

Mantis 同样是一款开源的软件缺陷管理工具，是一个基于 PHP 技术的轻量级缺陷跟踪系统，其功能与商用的 JIRA 系统类似，都是以 Web 操作的形式来提供项目管理及缺陷跟踪服务。Mantis 在功能上可能没有 JIRA 那么专业，界面也没有 JIRA 漂亮，但在实用性上足以满足中小型项目的缺陷管理及跟踪。Mantis 包括客户端浏览器、Web 服务器和数据库服务器。当然，Web 服务器和数据库服务器也可以是同一台主机。重要的是它是开源的，不需要付任何费用。不过 Mantis 目前的版本还存在一些问题，期待在今后的版本中能够得以完善。

1.2.1　Mantis 功能介绍

Mantis 基于 PHP+MySQL，可以运行于 Windows/UNIX 平台上。作为一个 Bug 管理系统，其适用性是否符合实际工作的需要是至关重要的。Mantis 基本可以满足 Bug 管理日

常流程。而且，Mantis 是 B/S 架构的 Web 系统，如果今后有需要，还可以配置到 Internet 上，实现异地 Bug 管理。在 Mantis 系统中，有如下几种角色：管理员、经理、开发人员、修改人员、报告人员、查看人员。每个角色所拥有的权限是不一样的，从大到小依次排列是：管理员→经理→开发人员→修改人员→报告人员→查看人员。

Mantis 的特点是免费、简洁灵活，B/S 结构的 Web 系统比较适合分布式协作开发和测试。关于 Mantis 的详细信息和技术支持，可访问 http://www.mantisbt.net/。

1. Mantis 的基本特征

(1) 个人可定制的 E-mail 通知功能，每个用户可根据自身的工作特点而只订阅相关的缺陷状态邮件。

(2) 支持多项目、多语言。

(3) 权限设置灵活，不同角色有不同权限，每个项目可设为公开或私有状态，每个缺陷也可设为公开或私有状态，每个缺陷可以在不同项目间移动。

(4) 主页可发布项目相关新闻，方便信息传播。

(5) 方便的缺陷关联功能。除重复缺陷外，每个缺陷都可以链接到其他相关缺陷。

(6) 缺陷报告可打印或输出为 CSV 格式。支持可定制的报表输出，可定制用户输入域。

(7) 有各种缺陷趋势图和柱状图，为项目状态分析提供依据，如果不满足要求，则可以把数据输出到 Excel 中进一步分析。

(8) 流程定制不够方便，但该流程可满足一般的缺陷跟踪。

(9) 可以实现与 CVS 的集成，即实现缺陷和 CVS 仓库中的文件相关联。

(10) 可以对历史缺陷进行检索。

2. Mantis 系统中缺陷状态的转换

缺陷状态是描述软件缺陷处理过程所处阶段的一个重要属性。对应于不同的状态，软件测试人员能确定对该问题的处理已经进展到什么阶段，还需要进行哪些工作，需要哪些人员的参与等信息。缺陷跟踪系统的状态比较复杂，这也是缺陷管理中的难点。在缺陷跟踪管理过程中，将缺陷记录划分为不同的阶段、不同的状态来进行标记。Mantis 系统将缺陷的处理状态分为 New、Active、Invalid、Later、Resolve、Reopen、Closed 7 种，如图 1-1 所示。

(1) 一个新的缺陷被提交，即为 New。

(2) Active，刚提交的缺陷，在被项目经理确认并分发给研发人员修改前所处的状态。

(3) Invalid，已提交的缺陷在当前版本中已不是问题或不需要修改。

(4) Later，提交的缺陷在当前研发阶段无法对其进行修改。

(5) Resolve，经软件工程师修改或给出相关意见后，等待测试人员验证时所处的状态。

(6) Reopen，已经关闭的缺陷重新出现，测试人员将其状态设置为 Reopen，分发缺陷时的操作与状态为 New 时的类似。

(7) Closed，最终修改正确或不正确的缺陷报告，经过验证或项目经理同意后，可以关闭。处于关闭状态的缺陷报告可表现为已改正、符合设计、不能重现、不能改正、由报告人撤回。

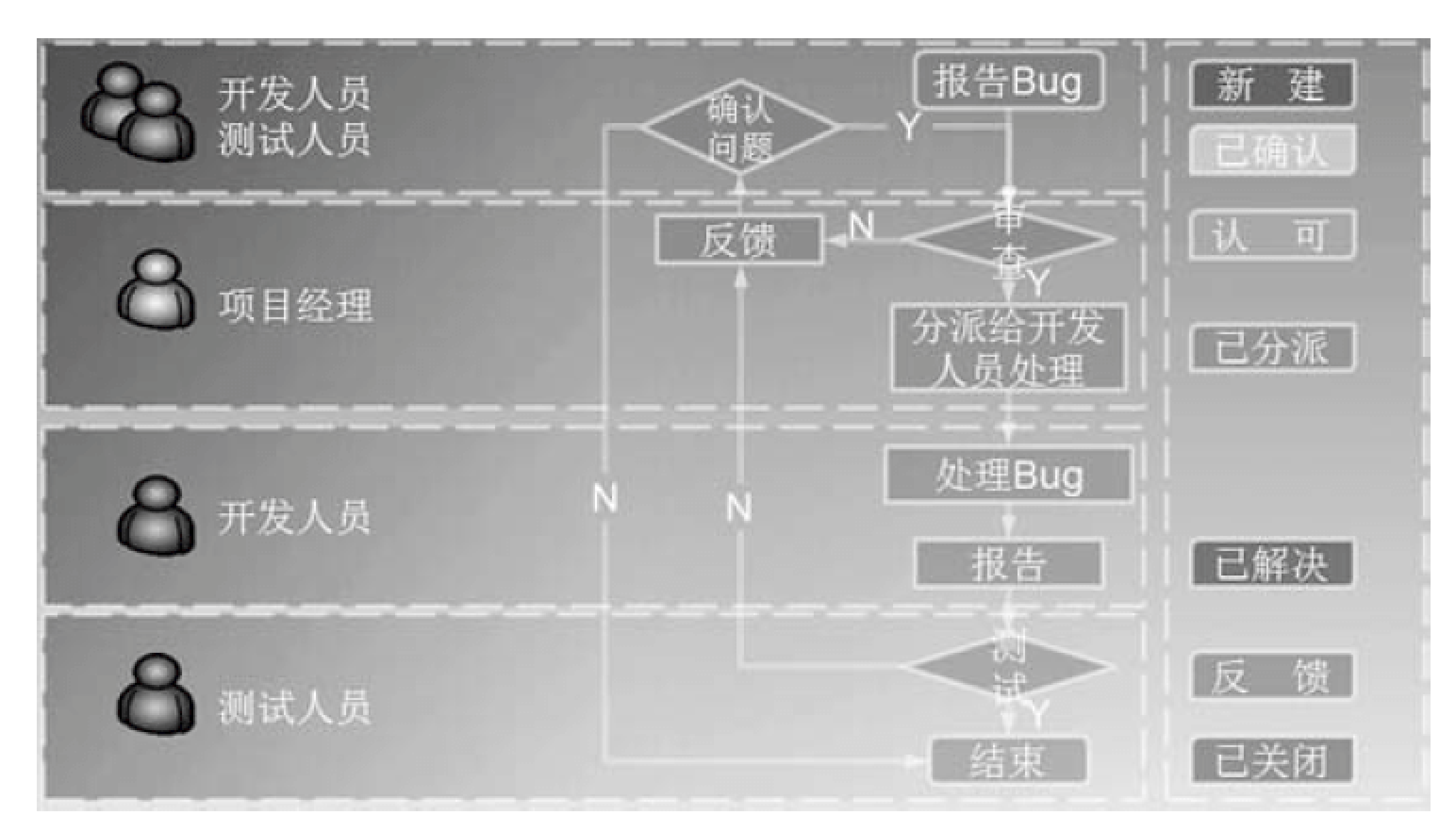


图 1-1　Mantis 缺陷状态转换图

3. Mantis 用户角色及权限的管理

在一个测试项目中，存在各种不同的身份，比如项目经理、测试经理、开发经理、程序员、测试员等。不同身份的用户使用系统时可以执行的操作理应是不同的，例如不能让一个测试员来进行用户分工的工作。另一方面权限的要求是以对象为中心的，比如对于缺陷这个对象，它的填报信息只能由填报该缺陷的测试员来修改和维护，其他的任何人都不能具有同等的操作权限。

Mantis 中用户角色和登录权限及方式如表 1-1 所示。

表 1-1　Mantis 中用户角色和登录权限及方式

用户	权限	工作范围
管理员 (administrator)	高	管理和维护整个系统
项目经理 (manager)	↓	对整个项目进行管理
开发人员 (developer)	↓	负责整个软件的开发
修改人员 (updater)	↓	负责修改Issue
报告人员 (reporter)	↓	负责提交Bug报告
查看人员 (viewer)	低	查看Bug流程及情况

4. Mantis 的软件缺陷属性的定义

软件缺陷是按照能准确发现缺陷目标进行分类的，分类之间应无重叠，分类体系应覆盖所有的缺陷类型；要与软件生命周期相结合。传统的分类方法可分为按照缺陷的来源和

缺陷错误性质这两种。如 Putnam 等人提出的分类方法和正交缺陷分类法以及 IEEE 制定的软件异常分类标准等。正交缺陷分类法定义的软件缺陷的 13 个属性在 Mantis 中得到了实现。

(1) 缺陷编号：缺陷的唯一标识。

(2) 模块信息：缺陷涉及的模块信息，包括模块名称、缺陷处理负责人、模块版本。

(3) 测试版本：描述的是该缺陷发现的测试版本号。

(4) 对应用例编号：发现该缺陷时运行的测试用例编号，通过该编号可以建立起测试用例和缺陷之间的联系。

(5) 缺陷状态：缺陷的即时状态，如 New、Active、Invalid、Later、Resolve、Reopen、Closed 等。

(6) 持有人：描述缺陷当前由谁负责，如果这个持有人是程序员，那么这个缺陷正在被修改；而如果持有人是测试员，则这个缺陷正在等待被确证。

(7) 报告者：报告缺陷的测试人员的编号或用户名。

(8) 报告日期：缺陷填报的日期。

(9) 重现性：可重现或不可重现。

(10) 重现步骤：和测试用例相关，描述的是发现这个缺陷的步骤。

(11) 严重等级：可定制，默认为 4 级——Pl(致命)、P2(严重)、P3(一般)、P4(轻微)。

(12) 缺陷类型：可定制，默认分为功能缺陷、用户界面缺陷、边界值相关缺陷、初始化缺陷、计算缺陷、内存相关缺陷、硬件相关缺陷、文档缺陷。

(13) 缺陷优先级(报告者)：可定制，默认分为必须修复、立即修复、应该修复、考虑修复。

5. Mantis 的功能介绍

1) 多项目管理

在系统页面上单击 Manage|Manage Projects，可以进入项目管理界面。上面显示了已创建的项目列表，单击 Create New Project，可进入新建项目页面。可以设定新项目当前的状态，项目状态有 development、release、stable 和 obsolete 这几种。在已建项目列表中可以修改项目数据，包括修改项目状态(将项目修改为公开或私有)，添加和修改子项目，为该项目添加和修改 Categories，添加和修改项目发布版本，定义项目可使用的用户自定义域，添加和修改该项目用户及其权限属性。

2) 问题录入

在系统界面单击 Report Issue，可进入问题录入界面。如果在单击前，右上角项目选择为 All Project，那么在填报问题前需要先选择要填报的项目。可以选中 Make Default，这样在每次填报进入该界面时，所选择的就是默认项目了。在问题填报界面选择并输入 Category、Reproduciblity、Impact、Severity、Summary、Description、Additional Information 等信息，单击 Submit Report 按钮即可。在录入页面中还可以添加和上传附件。

3) 问题查询和关键词检索

在系统界面，单击 View Issues，可进入问题查询结果页面。在项目选择中，可以选择项目查看所属项目问题，单击查询结果区的字段名称，可以进行排序显示。页面上方区域是问题检索条件区，可以一览当前查询结果的查询条件，也可以单击每个查询条件以修改该查询条件选项。修改各查询条件参数，单击 Apply Filter 按钮即可。该查询界面每个查询条件只能定义单一值。如果需要定义多值查询，可以在查询结果界面单击 Advanced Filters，界面刷新后，单击某查询条件，便可以选择多个选项进行查询。在查询结果页面的查询条件区，可以在 search 文本框中输入所要查询问题信息中的关键词，单击 Apply Filter 按钮，即可显示含有该关键词的所有历史问题。可以将当前查询条件保存为过滤器，以便快速选择得到查询结果。在查询区中单击 Save Current Filter，可以命名并保存当前过滤器。若当前过滤器的查询条件与已有过滤器的相同，那么保存页面会提示“This particular query appears to already exist”。输入待保存的过滤器，保存即可。在查询页面单击 Manage filters，可以管理过滤器。

4) 问题更新

(1) 单击 Assign to 按钮，将问题安排给相关人员解决。

(2) 可以单击 Due to 按钮，添加问题责任人。

(3) 单击 Change Status to，修改问题状态。

(4) 单击 Monitor Issue，可以跟踪该问题。

(5) 单击 Create Clone，可以克隆一个新问题。

(6) 单击 Move Issue，可以将问题在不同项目间进行移动。

(7) 单击 Delete Issue，可以删除该问题。

(8) 也可以单击 My View 或查询结果页面中某条问题前的图标，进入问题详细页面。单击按钮可以直接下载问题的附件。也可以在系统菜单右侧输入问题编号，即可进入问题详细页面。单击 Update Issue，可以修改问题的属性数据。

5) 问题讨论

在问题详细页面的后面添加 Note 信息，以便将该问题的讨论、交互信息记录下来。讨论信息可以进行编辑、删除，也可以被修改为私有状态。

6) 问题关联关系

在问题详细页面，可以设置该问题与其他问题之间的关联关系。每个问题都可以链接到其他相关问题。链接的关系分为 related to、parent of、child of、duplicate、has duplicate 几种。可以对当前链接的问题进行删除，有关系冲突的可以设置最新的关联关系。对于存在父子关系的问题，如果子问题没有解决，则父问题的关联关系中会显示 Not all the children of this issue are yet resolved or closed，以提示子问题没有被全部解决。

对于子问题没有全部解决的父问题，如果要将其状态设置为解决或关闭，则会在设置状态页面的上方提示“ATTENTION. Not all the children of this issue are yet resolved or closed. Before resolving/closing a parent issue，all the issues related as child with this one should be

resolved or closed"。通过单击问题详细页面中 Relationships 区域中的 Relation Graph，可以查看该问题的关联关系图。单击 Dependency Graph，可以查看当前问题的依存关系图。在关联关系图和依存关系图中，当光标移动到各问题的 ID 方框时，会显示该问题 ID 的 Status 和 Summary。

7) 集成 CVS

当将 CVS 文档提交给 CVS 服务器时，在 log message 中添加 issue #nnnn，提交后，即可将该提交信息插入到 issue #nnnn 的 Note 中。单击该提交的文件版本链接，弹出 commit 前后版本比较信息页面。通过单击系统菜单 Docs|CVSWeb，可以浏览 CVS 仓库。

8) 个人显示和 E-mail 通知设定

个人可定制的 E-mail 通知功能，使得每个用户可根据自身的工作特点而只订阅相关的缺陷状态邮件。在系统菜单中单击 My Account，进入用户个人设定页面。可以在 My Account 选项中修改用户密码和用户邮件地址，在 Preferences 中设定默认设置，可以对不同问题状态设定是否接收 E-mail。还可以设定自己的系统界面语言，为了实现多语言使用，一般使用各对应语言的 UTF-8 选项，可以选择的有：

(1) english_utf8。

(2) chinese_simplified_utf8。

(3) chinese_tranditional_utf8。

(4) japanese_utf8。

在 Profiles 中可以设定 Platform、Operating System、Version 等。

9) 统计分析、报表生成和输出

在系统菜单中单击 Summary，以显示该项目下问题统计 Synthesis 情况，包括按 Project、Status、Date、Resolution、Severity、Category 等进行统计的结果。单击 Summary 表上方的图表按钮，分别有 Per state、Per severity、Per impact、Per category 和 Per resolution 的统计表。后面仅列出了 Per state 的表截图。单击 Advanced Summary，可以显示总体统计图表，包括 Cumulative By Date 图。通过后台系统文件的设定，可以添加和修改统计图表。单击 Print Report，可打印当前项目下的问题。可以选择性地将问题导出至 Excel 或 Word 文件中，也可通过预览功能在 IE 中显示，并可另存为 HTML 文件。对于问题导出，还可以在问题查询结果页面中，通过单击 CSV Export，将问题导出为 CSV 文档。在问题查询结果页面单击 Print Report，可以进入打印报告页面。

10) 用户管理

使用管理员账户进入系统，单击系统菜单 Manage|Manage Users，进入用户一览页面。可以按用户 ID 的字母顺序筛选用户。可以单击各用户以修改其权限和信息，也可以单击 Prune Accounts 来阻止未登录的用户。单击 Create New Account 建立新账户时，可以选择是否激活该账户，也可以设定用户权限。用户权限包括 viewer、reporter、updater、developer、manager 和 administrator(角色可以定制)。权限可以在系统权限设置中进行控制。

11) 自定义域

通过单击系统菜单 Manage|Manage Custom Fields，用户可以自行添加和修改自定义域，添加数量没有限制。自定义域的类型有 String、Numeric、Float、Enumeration、Email、Checkbox、List、Multiselection List、Date 等。可以设置是否在报告、更新、解决、关闭页面中显示和必填，以及是否仅在高级查询条件页面中显示。

12) 系统设置

使用管理员权限进入系统，单击 Manage|Manage Configuration，进入系统设置页面。Permissions Report 页面显示了当前系统的权限分配情况。在 Workflow Thresholds 页面，可以设置不同角色权限。在 Workflow Transitions 页面，可以设置工作流。可以根据公司流程来进行定制。可以设定问题各状态的最低权限角色。

13) 新闻发布

新闻发布后，可以在系统菜单 Main 中进行显示，这样用户一进入系统就可以看到。

1.2.2　Mantis 应用环境建立

要安装运行 Mantis，有两种主流的环境配置可供选择：IIS+PHP+MySQL+Mantis 和 Apache+PHP+ MySQL+Mantis，下面介绍后一种。由于单个配置相当复杂，且容易出错，这里采用安装 XAMPP 来配置环境。

XAMPP（Apache+MySQL+PHP+Perl）是一个功能强大的软件集成包。这个软件包原来的名字是 LAMPP，但是为了避免误解，最新的几个版本就改名为 XAMPP 了。它可以在 Windows、Linux、Solaris、Mac OS X 等多种操作系统下安装使用，支持多语言：英文、简体中文、繁体中文、韩文、俄文、日文等。许多人通过他们自己的经验认识到安装 Apache 服务器是件不容易的事。如果想添加 MySQL、PHP 和 Perl，那就更难了。XAMPP 是一个易于安装且包含 MySQL、PHP 和 Perl 的 Apache 发行版。

1. 安装 XAMPP

这个安装比较简单，直接安装就行，这里安装的是 XAMPP Windows 1.8.3 版本。下载网址为 http://www.apachefriends.org/zh_cn/xampp.html。

安装完之后，打开控制面板，如图 1-2 所示，表明 Apache 和 MySQL 正在运行，单击图 1-2 中 Apache 一行中的 Admin 按钮，弹出 XAMPP 页面，选择中文后，单击左边的“安全”，出现 XAMPP 安全页面，如图 1-3 所示。

修改 MySQL 中的 root 密码为“root”，如图 1-4 所示。

2. 安装 Mantis

这里选择安装最新的 Mantis 发布版本 1.2.15，登录 http://localhost/phpmyadmin，用户名和密码均为 root，创建数据库 mantis，如图 1-5 所示。

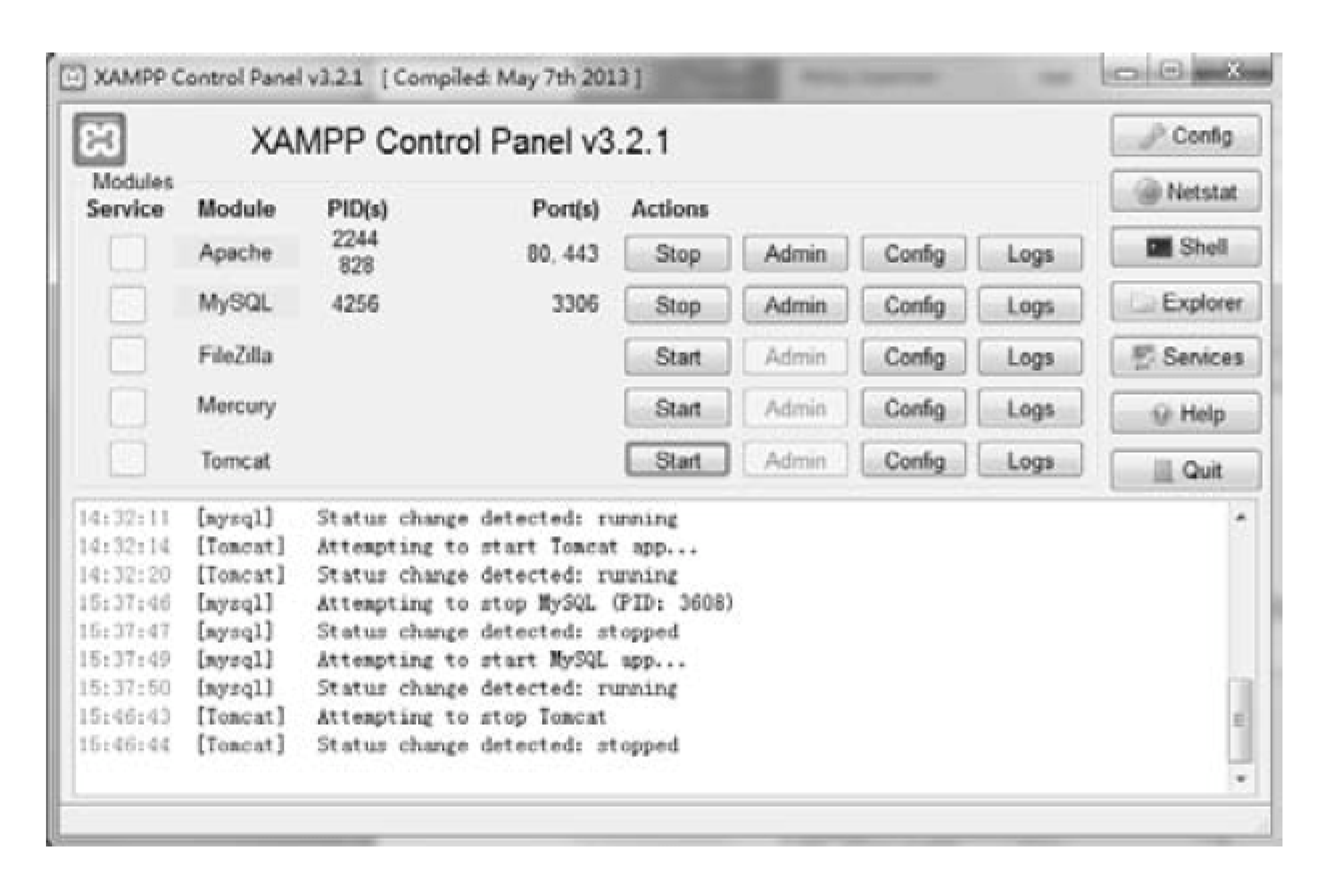


图 1-2　XAMPP 控制面板

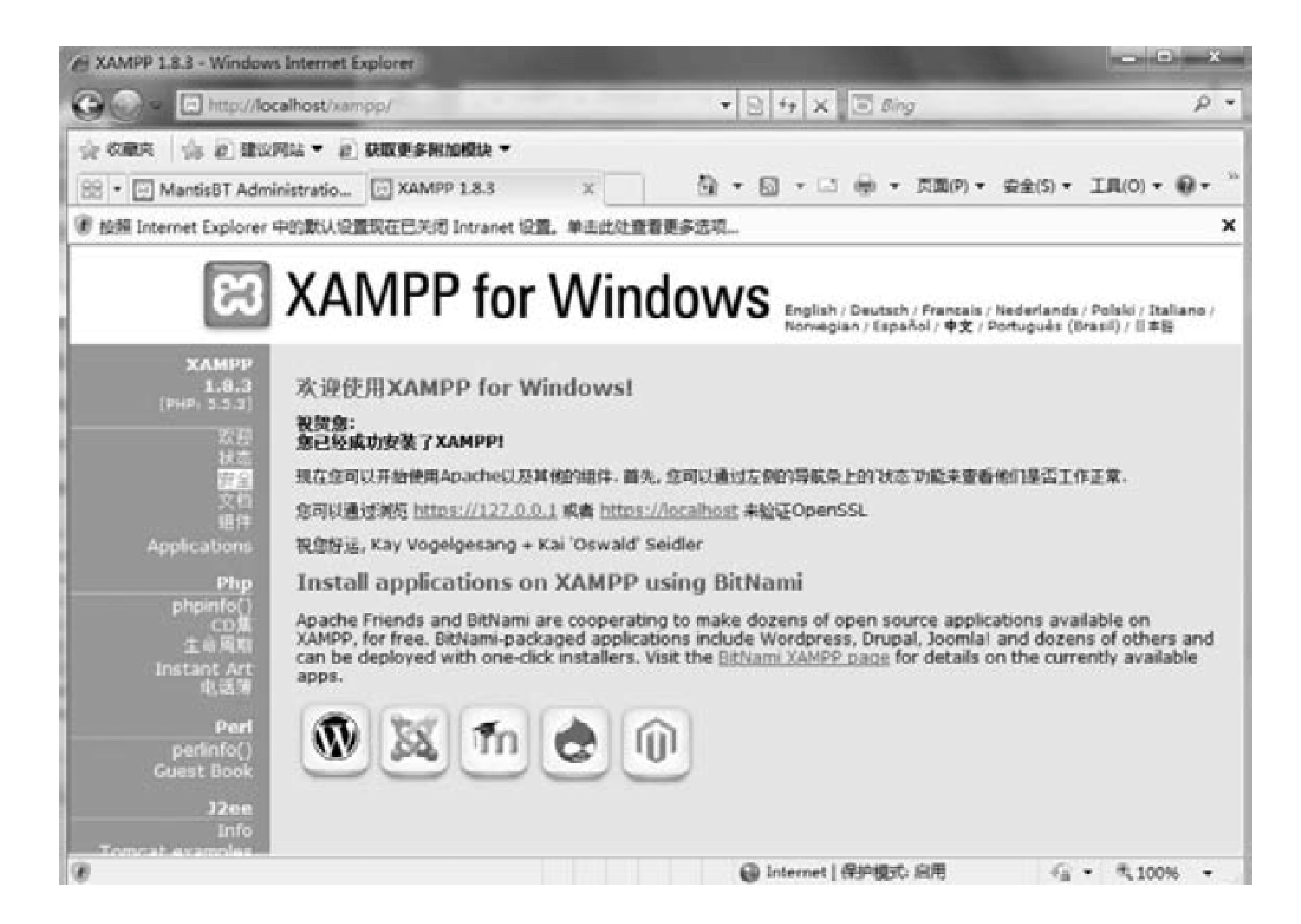


图 1-3　XAMPP 安全页面

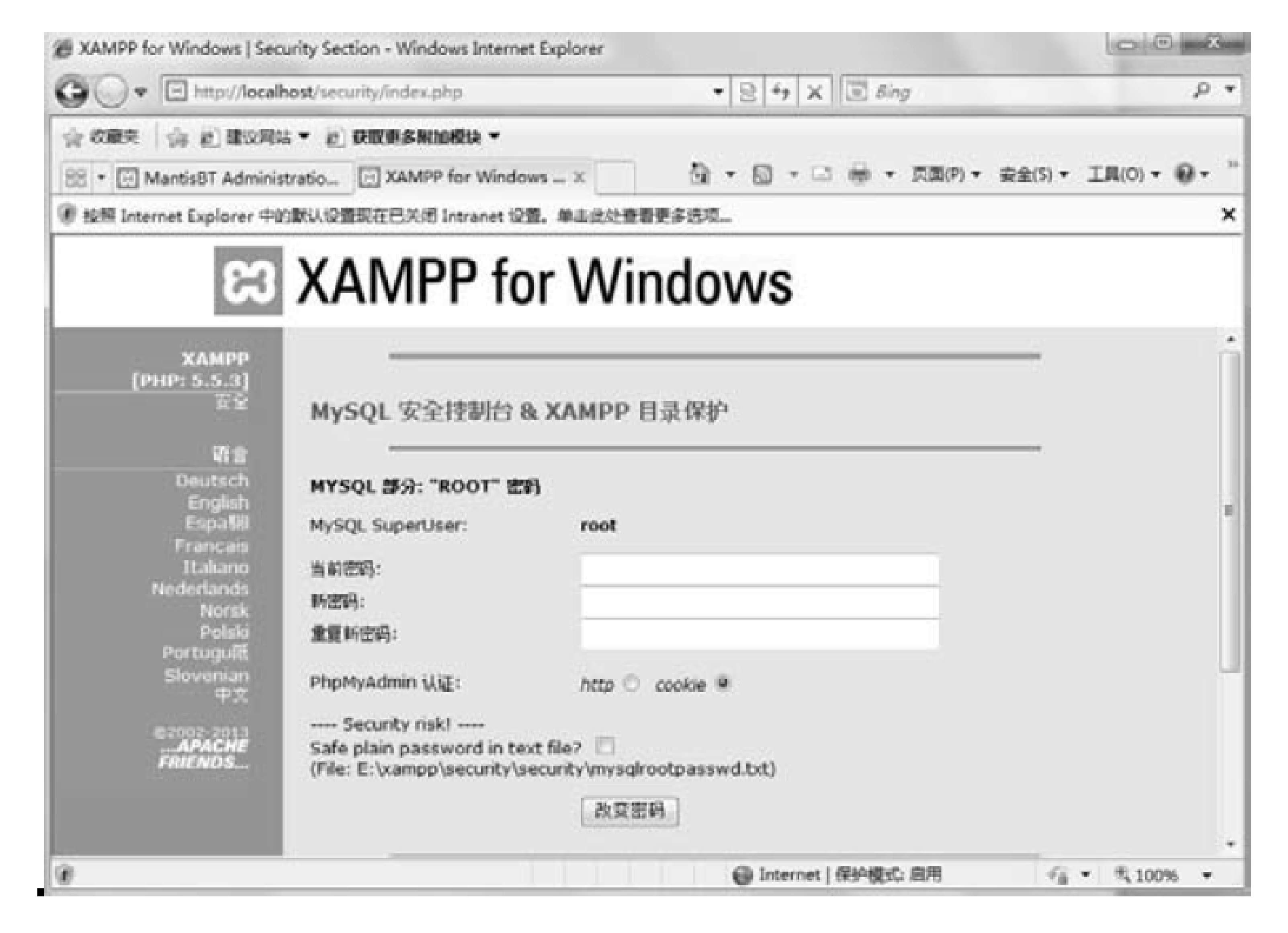


图 1-4　MySQL 密码修改

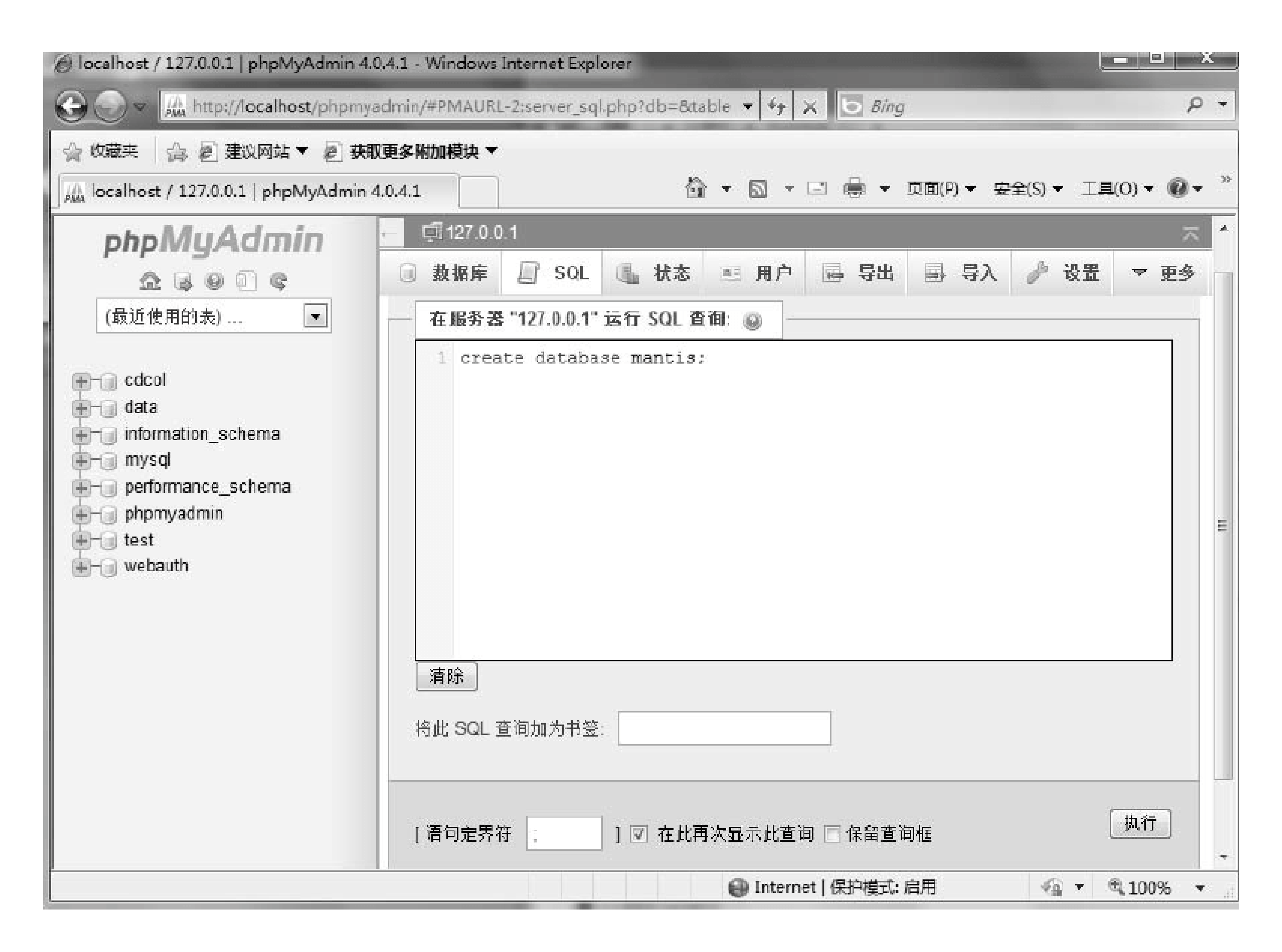


图 1-5　创建数据库 mantis

把 mantis 1.2.15 解压到 E:\xampp\htdocs\目录下，并改名为 mantis。

打开 IE，输入“http://localhost/mantis”，即可进入安装界面，如图 1-6 所示。

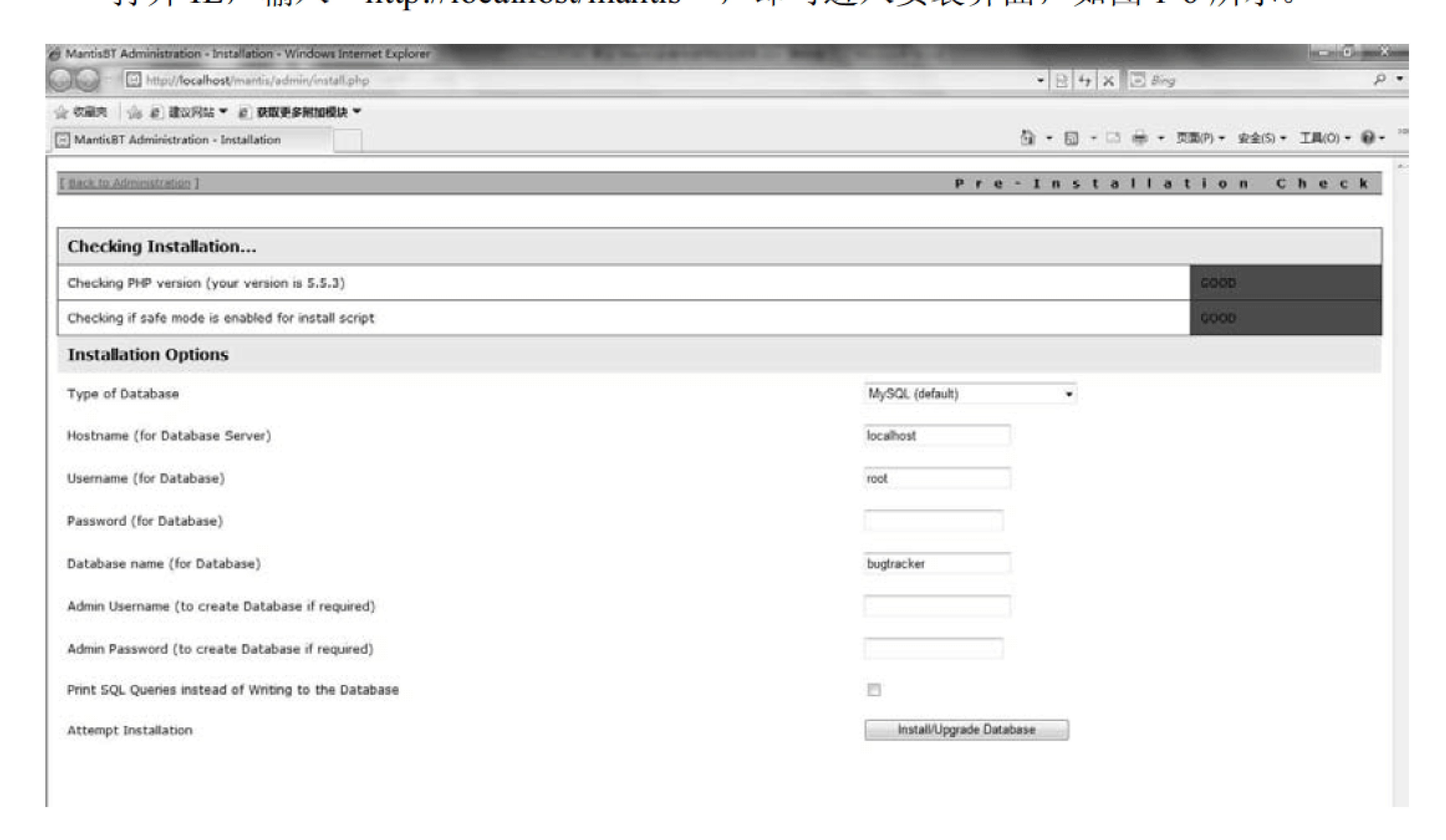


图 1-6　Mantis 安装界面

(1) 在安装界面，除了 Database name 填 mantis 外，上下两个 user name 和 password 都填 root。如果安装成功，后面的状态栏为全绿，如图 1-7 所示。

(2) 打开 IE，输入“http://localhost/mantis”，可以看到 Mantis 的登录页面，如图 1-8 所示。初始用户可以使用默认用户名 administrator 和密码 root 登录进去，进行管理设置。

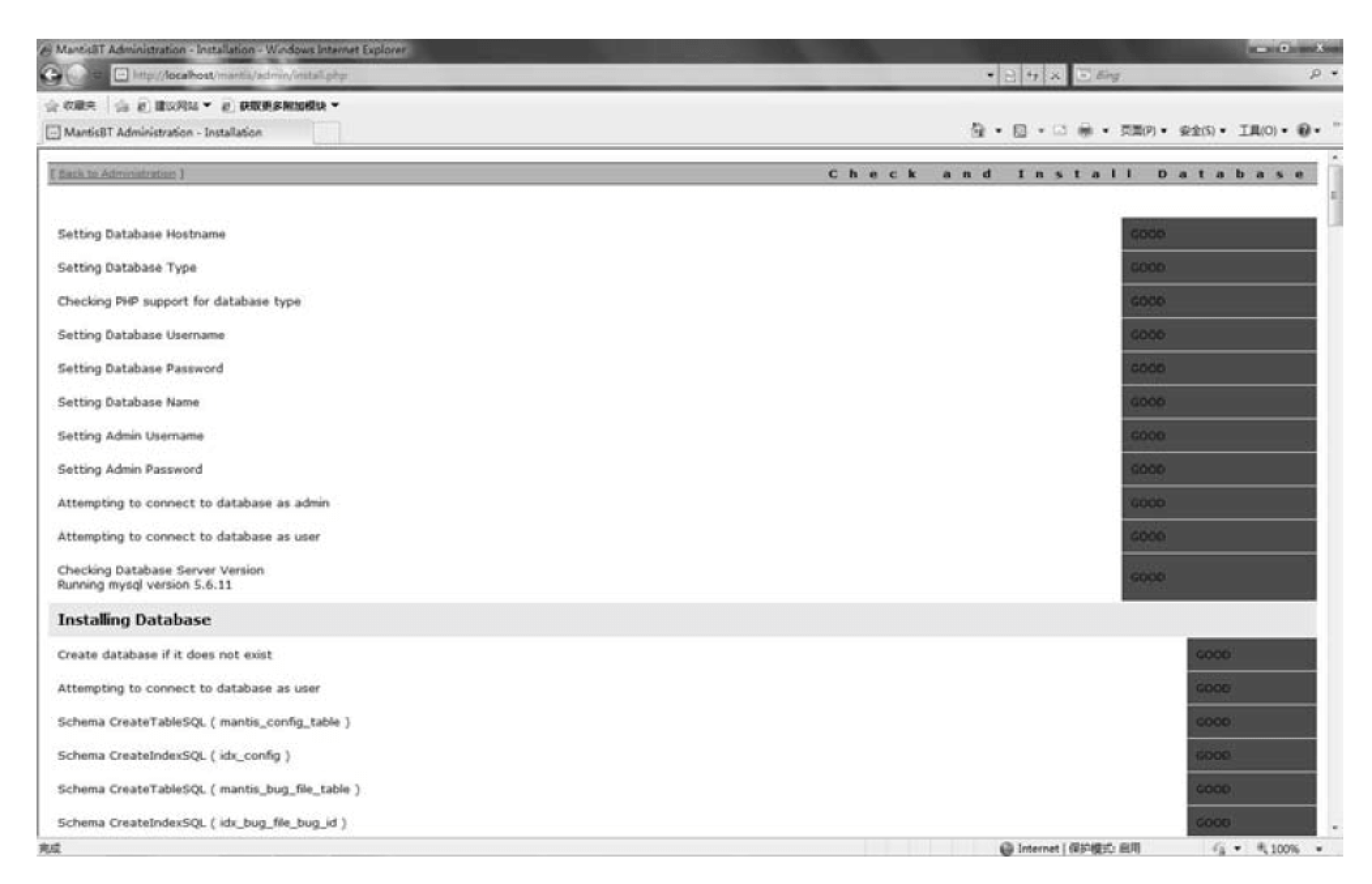


图 1-7　Mantis 安装成功



Login	
Username	
Password	
Remember my login in this browser	☐
Secure Session	☑ Only allow your session to be used from this IP address.
	Login

[Signup for a new account] [Lost your password?]

Warning: You should disable the default 'administrator' account or change its password.

Warning: Admin directory should be removed.

图 1-8　Mantis 登录界面

3. Mantis 的其他设置

Mantis 的设置是这样保存的：在 config_defaults_inc.php 中保留 Mantis 的默认设置，用户自己的设置信息保存在 config_inc.php 中，如果某个选项在 config_inc.php 中已设置，则系统使用 config_inc.php 中的设置，否则使用 config_defaults_inc.php 中的系统默认设置。

config_inc.php.sample 是 Mantis 给出的一个用户设置文件示例。sample 中给出的一些设置是一定需要修改的，比如 MySQL 数据库的连接参数、管理员的邮箱；其他的则要根据实际情况进行修改。对 config_inc.php 文件中的设置很简单，各个参数的意义参见 config_

defaults_inc.php，其对每个参数都有详细的解释。

下面是其中的一些自定义参数，一些参数的设置请参照下面的内容。

```
$g_use_iis = ON;                                    # 使用 IIS
$g_show_version = OFF;                              # 不在页面下部显示 Mantis 的版本号
$g_default_language = 'chinese_simplified';         # 默认语言为简体中文
$g_show_project_menu_bar = ON;                      # 显示项目选择栏
$g_show_queries_count = OFF;                        # 在页脚处不显示执行的查询次数
$g_default_new_account_access_level = DEVELOPER;    # 默认用户级别
$g_use_jpgraph = ON;                                # 使用图形报表
$g_jpgraph_path = 'C:/PHP/includes/JPGraph/src/';   # JPGraph 路径
$g_window_title = 'Mantis Bug 跟踪管理系统';        # 浏览器标题
$g_page_title = 'Mantis Bug 跟踪管理系统';          # 页面标题栏
$g_enable_email_notification = ON;                  # 开通邮件通知
$g_smtp_host = 'smtp.mail.net';                     # SMTP 服务器
$g_smtp_username = 'mailuser';                      # 邮箱登录用户名
$g_smtp_password = 'mailpwd';                       # 邮箱登录密码
$g_use_phpMailer = ON;                              # 使用 PHPMailer 发送邮件
$g_phpMailer_path = 'C:/PHP/includes/PHPMailer/';   # PHPMailer 的存放路径
$g_phpMailer_method = 2;                            # PHPMailer 以 SMTP 方式发送 E-mail
$g_file_upload_ftp_server = 'ftp.yourftp.com';      # 上传文件 FTP
$g_file_upload_ftp_user = 'ftpuser';                # FTP 登录用户名
$g_file_upload_ftp_pass = 'ftppwd';                 # FTP 登录密码
$g_short_date_format = 'Y-m-d';                     # 短日期格式，Y 大写表示以 4 位表示年数
$g_normal_date_format = 'Y-m-d H:i';                # 普通日期格式
$g_complete_date_format = 'Y-m-d H:i:s';            # 完整日期格式
```

1) 设置 Mantis 邮件服务

首先修改 E:\xampp\php 目录下的 php.ini 文件，查找 SMTP，将 SMTP = localhost 改为发件服务器（这里以 163 的邮箱为例），如 SMTP = smtp.163.com；查找 sendmail_from，并修改为 sendmail_from = xxx@163.com（设置邮箱地址全称）。

然后在 E:\xampp\htdocs\mantis 目录下的 config_inc.php 文件中，添加：

```
$g_phpMailer_method = 2;
$g_smtp_host='smtp.163.com';
$g_smtp_username='xxx';                             //邮箱用户名
$g_smtp_password='yyy';                             //邮箱密码
```

最后在同目录下的 config_default_inc.php 文件中做如下修改：

```
$g_phpMailer_method = PHPMAILER_METHOD_SMTP;
$g_smtp_host='smtp.163.com';
$g_administrator_email='xxx@163.com';               //邮箱地址全称
```

$g_webmaster_email='xxx@163.com';

$g_from_email='xxx@163.com';

$g_return_path_email='xxx@163.com';

2) 设置图形报表

默认情况下，Mantis 的图形报表是关闭的，需要安装图形报表模块才能打开图形报表。

下载 JpGraph，从 http://jpgraph.net/download 下载 JpGraph 的安装文件，目前最高版本是 3.5.0b1。

将 jpgraph-3.5.0b1.tar.gz 解压缩，把其中的子目录 src 复制到 mantis\library 目录下，并改名为 jpgraph。

修改 E:\xampp\php 目录下的 php.ini 文件，将“;extension=php_gd2.dll”前面的分号删除，确保 PHP 有加载 JpGraph 使用的动态库。

安装插件：登录 Mantis（管理员权限），管理→插件管理→安装 MantisGraph 1.0 插件，如图 1-9 所示。



图 1-9　安装 MantisGraph 插件

修改 MantisGraph 插件配置，将要使用的图形库修改为 JpGraph，JpGraph 库系统路径设置为对应的 JpGraph 路径，例如 E:\xampp\htdocs\mantis\library\jpgraph，如图 1-10 所示。

单击 Update Configuration，现在再打开 Mantis 的统计页面，可以看到多了分别按状态等进行统计的图形报表，包括柱图、饼图和线图。

如果使用的界面语言是简体中文，那么将会看到图形中的汉字都是乱码，这是由于 Mantis 对 JpGraph 的编码设置不正确造成的。JpGraph 会自动将汉字转换为 UTF-8 编码，但是需要在调用 JpGraph 的时候对标题等设置字体，Mantis 没有做这个操作因此汉字显示出来都是乱码。解决方法如下。



图 1-10　MantisGraph 插件配置

(1) 修改 mantis\library\jpgraph\jpgraph_ttf.inc.php。

```
elseif( $aFF === FF_SIMSUN ) {
            // Do Chinese conversion
            if( $this->g2312 == null ) {
                include_once 'jpgraph_gb2312.php' ;
                $this->g2312 = new GB2312toUTF8();
            }
            return $this->g2312->gb2utf8($aTxt);
    }
```

将其改为：

```
/*elseif( $aFF === FF_SIMSUN ) {
            // Do Chinese conversion
            if( $this->g2312 == null ) {
                include_once 'jpgraph_gb2312.php' ;
                $this->g2312 = new GB2312toUTF8();
            }
            return $this->g2312->gb2utf8($aTxt);
      }*/
elseif( $aFF === FF_SIMSUN ) {
            return $aTxt;
}
```

(2) 修改 mantis\plugins\Mantisgrpah\pages\config.php。

① 增加字体 simsun。

```
$t_current_font_selected = array('arial' => false,
```

将其改为：

```
$t_current_font_selected = array('simsun' => false,   'arial' => false,
```

② 配置页面显示新增加的 simsun（宋体）。

```
<label><input type="radio" name="font" value="arial"<?php echo print_font_checked( 'arial' )?>/>
Arial</label><br />
```

将其改为：

```
<label><input type="radio" name="font" value="simsun"<?php echo print_font_checked( 'simsun' )?>/>宋
体</label><br />
<label><input type="radio" name="font" value="arial"<?php echo print_font_checked( 'arial' )?>/>
Arial</label><br />
```

注：因直接使用中文，为不显示为乱码，需要把该代码文件转换成 UTF-8 编码格式，文件另存为时选择即可。

(3) 修改 mantis\plugins\MantisGraph\pages\config_edit.php。

```
if ( plugin_config_get( 'font' ) != $f_font ) {
    switch ( $f_font ) {
        case 'arial':
```

将其改为：

```
if ( plugin_config_get( 'font' ) != $f_font ) {
    switch ( $f_font ) {
        case 'simsun':
         case 'arial':
```

(4) 修改 mantis\plugins\MantisGraph\core\graph_api.php。

```
$t_font_map = array('arial' => FF_ARIAL,
```

将其改为：

```
$t_font_map = array('simsun' => FF_SIMSUN, 'arial' => FF_ARIAL,
```

(5) 修改 MantisGraph 插件配置，管理→插件管理→Mantis 图表 1.0，编辑配置，修改字体为“宋体”，如图 1-11 所示。

这样，图形报表就可以显示中文了。

1.2.3　Mantis 应用流程

Mantis 系统中，分别有如下几个角色：管理员、经理、开发人员、修改人员、报告人



图 1-11　修改字体为“宋体”

员、查看人员。每个角色所拥有的权限是不一样的，权限从大到小依次排列是：管理员→经理→开发人员→修改人员→报告人员→查看人员。

首先来看主页，在主页面中可以看到以下信息。

(1) 指派给登录用户且还未解决的 Issue 数目，单击该数目链接，就可以进入 Issue 列表，直接对这些 Issue 进行操作。

(2) 由登录用户报告还未解决的 Issue 数目。

(3) 最后一次登录该系统的时间。

切换项目：单击界面中右上角的下拉式菜单来切换所选项目。

跳转到该 Issue 编号：根据 Issue 编号可以进行查询，直接进入该 Issue 的详细信息界面，进行相应操作。

转向其他操作界面：单击主页面上方的菜单栏，即可进入相应的操作界面，如图 1-12 所示。

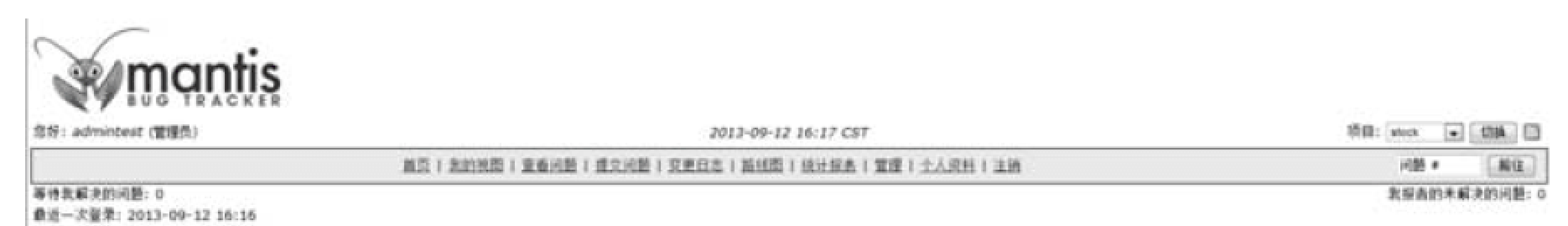


图 1-12　Mantis 的有关操作界面

下面将详细说明各个角色在系统中的具体权限和工作职责。

1. 管理员

管理员是管理整个系统运作的工作人员，他不仅是整个系统操作流程中权限最高的工作人员，而且还可以对项目和用户账户进行创建和管理等，下面将详细说明。管理员登录系统之后，可以先进入自己的主界面，然后再根据工作要求，选择页面上方的菜单栏来进入相应的界面。

1) 我的视图

“我的视图”（My View）界面如图 1-13 所示。

Bug 根据其工作状态被分类成几个表格来显示，符合这些工作状态的 Bug 都被一一罗列。

(1) 分派给我的，assigned(尚未解决)。

(2) 未分派的，unassigned。

(3) 我报告的，reported by me。

(4) 已解决的，resolved。

(5) 最近修改，recently modified。

(6) 我监视的，monitored by me。

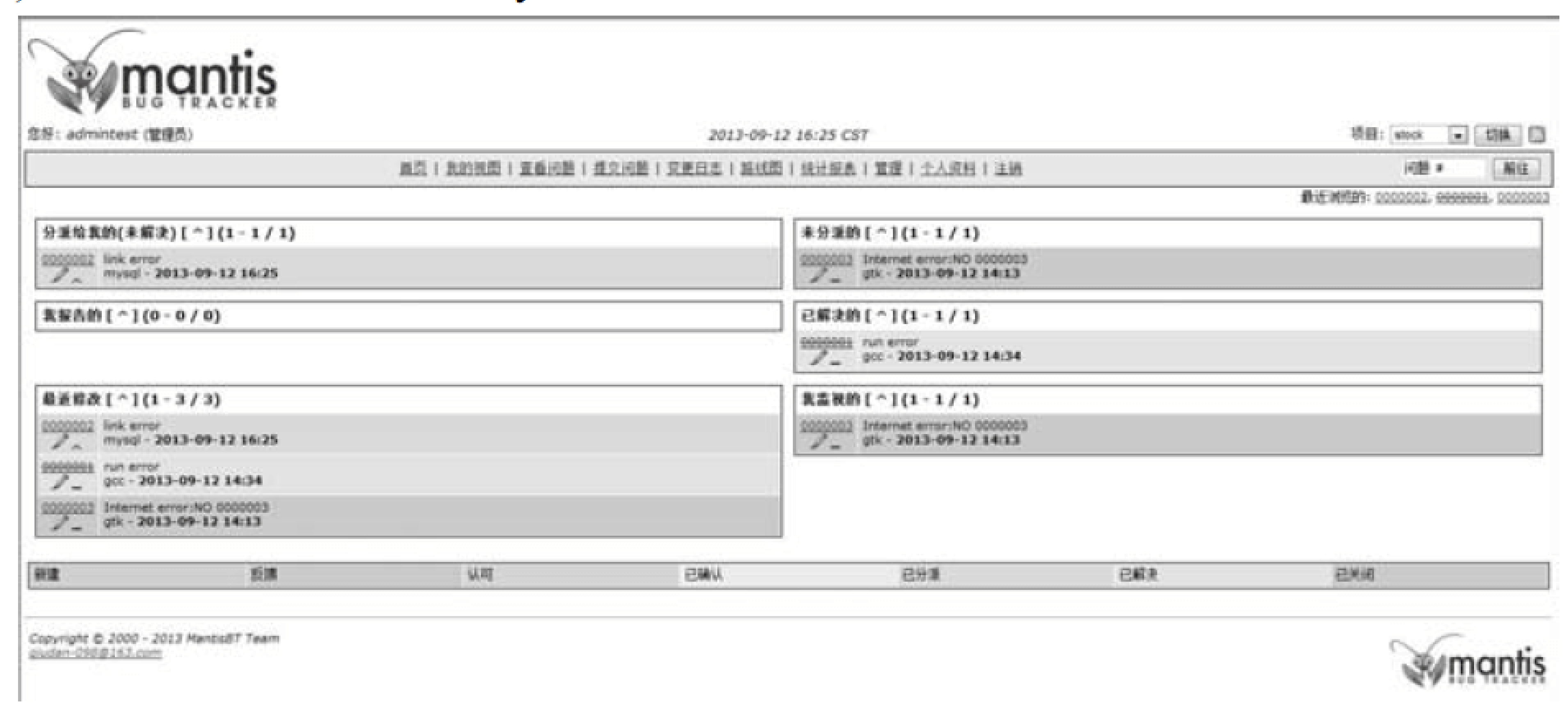


图 1-13　Mantis 的视图界面

2) 查看问题

如图 1-14 所示为查看问题（View Issues）界面。

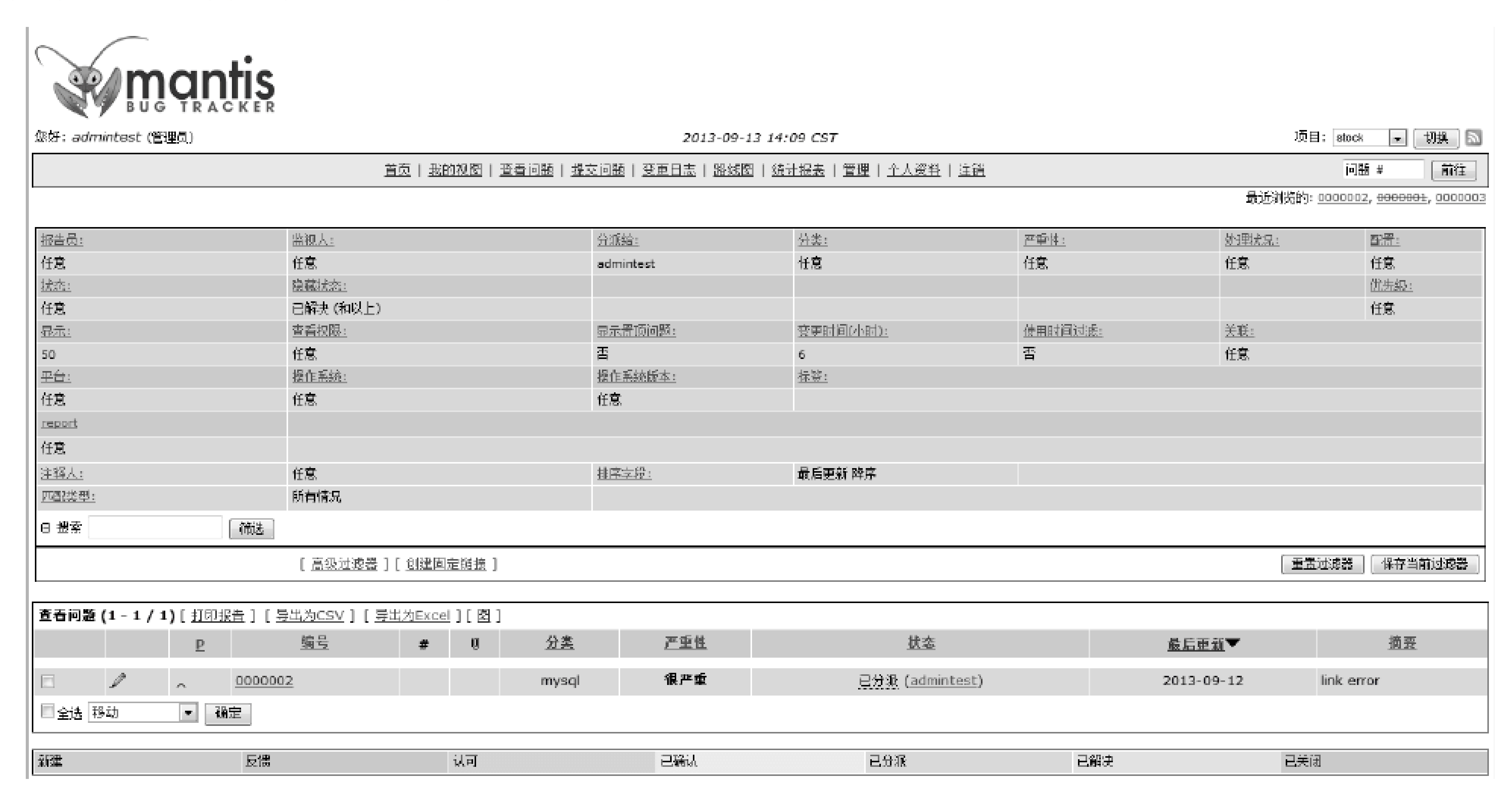


图 1-14　Mantis 的查看问题界面

单击页面中的问题编号，可进入该问题的详细页面，并对该问题进行修改，如图 1-15 所示。

(1)“编辑(Edit)”：修改问题的各项基本属性，并添加注释。

(2)“分派(Assign To)”：将问题分派给某个开发人员处理，分派之后系统将自动向被分派人发送邮件通知，被分派人打开 Mantis 之后将在“我的视图”页看到被分派的问题。

(3)“状态改为(Change Status to)”：这里是指问题状态的转变，分为 6 个层次——新建、反馈、认可、已确认、已解决和已关闭。这是 Mantis 比较重要的一个功能，问题的每次变动都会发生状态的改变，以此来标记问题的处理情况。



图 1-15　Mantis 的问题详细界面

(4)“监视问题(Monitor)”：单击此按钮后，用户就可以对该问题进行监视，也就是说只要该问题有改动，系统就会自动发邮件通知到本人。这在“我的视图”页也可以体现出来。

(5)“创建子项问题(Clone)”：可以创建该问题的子项问题。

(6)“移动问题(Move)”：可以将该问题移动到别的项目中(需要相应的权限)。

(7)“删除问题(Delete)”：删除无用的问题，已处理完毕的问题建议不必删除，关闭即可，以保留问题记录。

(8)“关系(Relationships)”：可以指定问题之间的关联关系，具体关联方式见下拉菜单。

(9)“上传文件(Upload File)”：可以上传与问题相关的文件，大小暂时限制为 5MB。

(10)“问题历史(Issue History)”：此项为问题处理的历史记录。

3) 提交问题

如图 1-16 所示，进入 Report Issue 界面，可以看到一个提交 Bug 的表单，根据具体情况填写后提交即可。在提交报告时请注意，带*号的是必填项。页面还提供了文件上传功能，只要是大小小于 2MB 的文件都允许上传，支持 doc、xls、zip 等格式的文件。这样在报告 Bug 的时候，就可以上传相关的文件，为 Bug 的解决提供更多的信息。全部填写完毕之后就可以单击“提交报告”按钮来提交报告，之后系统会提示用户操作成功。返回“我的视图”界面，就可以看到新提交的报告。

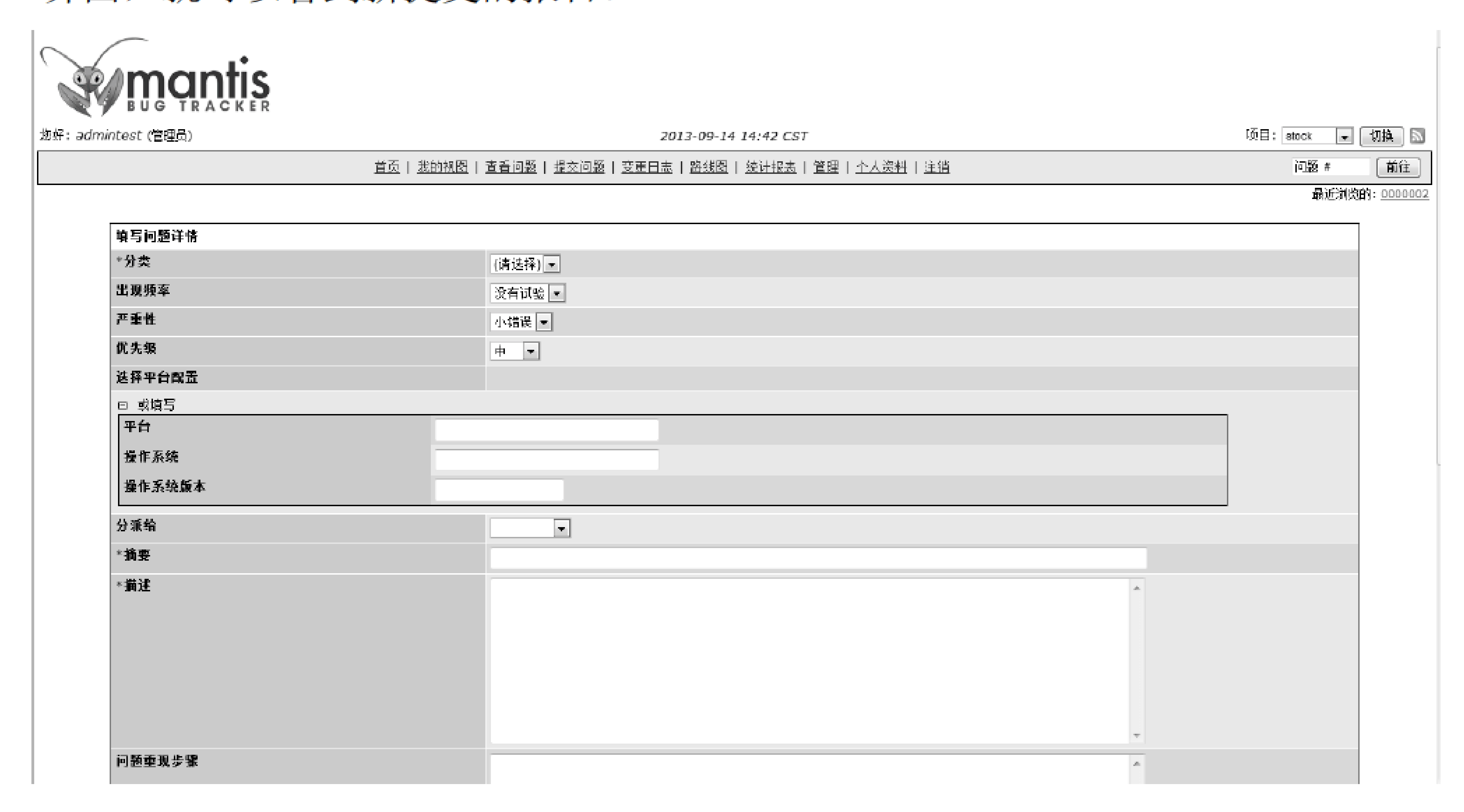


图 1-16　Mantis 的问题提交界面

4) 修改日志

修改日志(Change Log)界面如图 1-17 所示。单击菜单栏中的“变更日志”项，是已经修改好了问题的日志，需要给项目添加版本号，并且在提交/解决问题时都指定了相应的版本号才会显示。



图 1-17　Mantis 的日志修改界面

5) 路线图

路线图(Roadmap)界面如图 1-18 所示。单击菜单栏中的“路线图”，如果开始没有指定项目的情况下，则首先进入项目选择界面，如果指定了默认值，则直接进入默认项目的

路线图。其实项目的路线图就相当于一个 Bug 的日志信息，如图所示，页面列出了该项目下已解决的 Bug 编号、所属组别、Bug 摘要以及该项目产品的版本号变化，单击 Bug 编号还可进入其详细信息页面。



图 1-18　Mantis 的路线图

6) 个人资料

个人资料(My Account)界面中可设置个人信息，包括密码、邮件、姓名；更改个人设置，可设置邮件通知的紧急程度级别等，根据个人需要和喜好来设置接收邮件通知的级别；管理平台配置，包括硬件平台；显示操作系统、版本等信息，如图 1-19 所示。



图 1-19　Mantis 的个人资料设置界面

7) 统计报表

统计报表(Summary)界面如图 1-20 所示。单击“统计报表”项，将会出现一个包含所有 Issue 报告的综合报表，页面上还提供了“打印报告”和“先进的摘要”的功能链接。在这个综合报表中，按照 Bug 报告详细资料中的项目，将所有的报告按照不同的分类进行了统计。这个统计报表有助于管理员及经理很好地掌握 Bug 报告处理的进度，而且很容易就能把没有解决的问题与该问题的负责人、监视人联系起来，提高了工作效率。这个界面中还提供了更多按不同要求分类的统计图表，如按状态统计、按优先级统计、按严重性统计、按项目分类统计和按处理状况统计。分别单击这 5 项，即可得到相应的统计图。单击“先进的摘要”，可进入刚才所显示数据的一个图形化报表界面，更便于查阅和对比，如

图 1-20 所示。如果需要，还可以单击界面上方的“打印报告”，将所有的 Bug 显示出来。如图 1-21 所示，在页面列出了需要导出打印的 Bug 列表，可以根据需要通过复选框选中需要打印的 Bug，根据需要单击图标，Bug 数据便相应地导出到该类型的文件里，实现打印输出的需求。

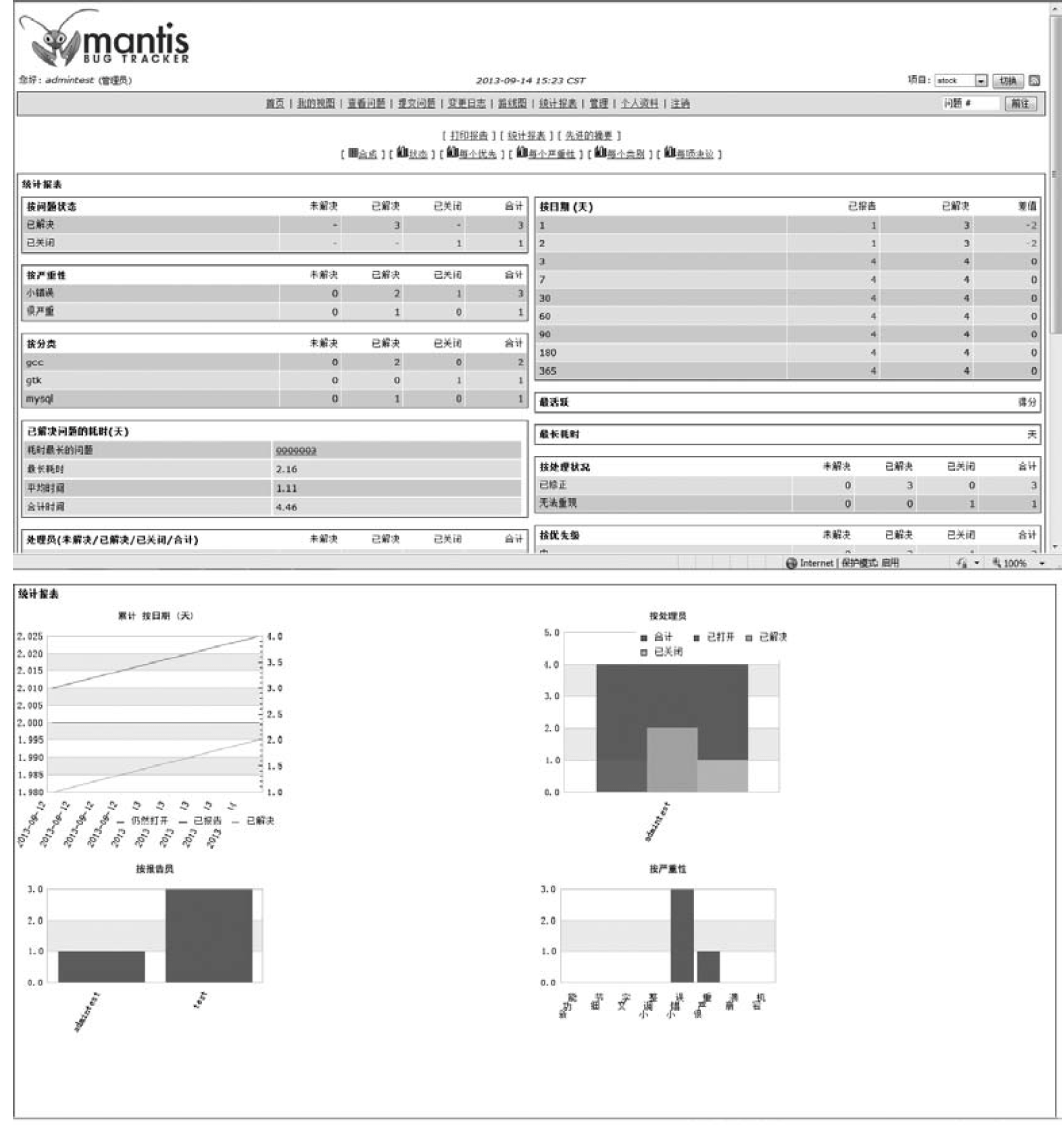


图 1-20　Mantis 的统计报表及统计图

MantisBT - stock

查看问题(1 - 1)

	P	编号	#	0	分类	严重性	状态	最后更新▼	摘要
☐	^	0000002	2		mysql	很严重	已分派 (admintest)	2013-09-14	link error

只显示选中内容

图 1-21　打印报告界面

8) 管理

管理(Manage)界面如图 1-22 所示。单击菜单栏的“管理”项，即可进入管理界面，管理界面包含用户管理、项目管理和自定义字段管理的管理部分。

(1) 用户管理为单击“管理”项时进入的默认页面。

新账号：显示一周之内添加到该项目的新用户。

从未登录：显示目前存在却至今从未登录过的用户，对此用户可以单击“清理账号”功能链接将其清除。

管理账号：在这里可以添加新用户和更新已有用户。单击 Create New Account，就可以添加新的用户，并指定其工作身份。单击现有账号名称，就可以对当前账号的资料进行更新，更新之后单击“重置”，系统就接受更新信息了。

(2) 单击“项目管理”，即可查看当前的所有项目。再单击“创建新项目”项，进入新项目创建页面，填写好项目资料后单击“添加项目”，新的项目就添加到系统中了。

注意：上传文件时如果不指定存放路径，则默认为系统路径。

(3) 单击现有项目列表中的项目名称，就可以看到该项目的具体情况列表，列表中包括各个项目的名称、状态、查看状态以及说明列属性。在这个页面上可以进行以下操作。

编辑项目：在这里经理可以对项目的名称、状态、查看状态、上传文件的存放路径以及说明等内容进行更新。

子项目：可以创建属于该项目的子项目，或者指定某个项目为该项目的子项目。

增加分类：填入类别名称，单击“增加分类”便可在当前项目里增加类别。

编辑分类：单击“编辑”，进入类别编辑页面，还可以将当前的类别分配给指定的工作人员，这样会在该项目下提交一个新 Bug 的时候，直接分派给该指定的工作人员处理。

版本：可以对已有的项目版本进行更新或者删除，也可以添加新的版本。

自定义字段：可以从已存在的自定义字段中选择出所需要的添加到该项目中的自定义字段里，也可以删除已添加的自定义字段，自定义字段添加入项目后，在“提交问题”单中会显示为必填字段。

添加用户至项目：将与项目相关的用户添加进来。

管理账号：对该项目中所有的相关人员的账号进行管理，可以删去那些在项目中不需要的账号。

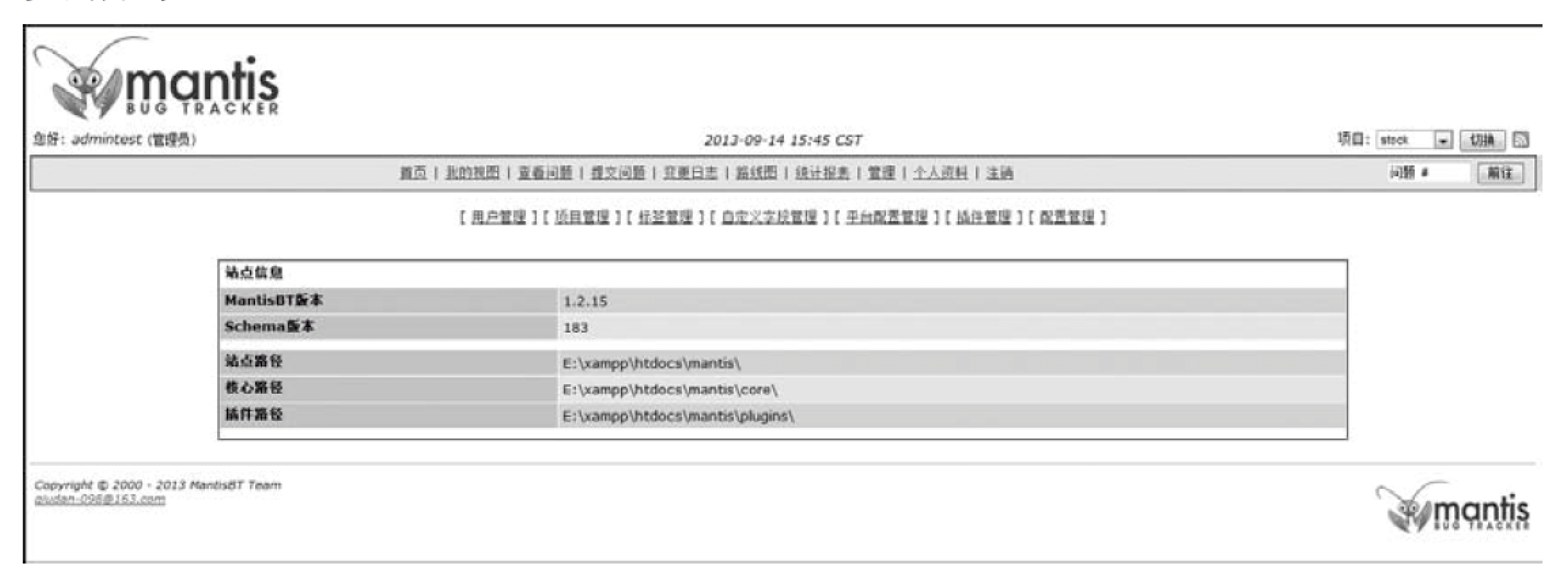


图 1-22　Mantis 的管理界面

(4) 自定义字段管理是用于在提交问题的时候，系统给予的填写项不满足实际需求，这样可以自行在这个功能页面里定义自己需要的字段，以便能更好地描述 Bug 的情况，而且在过滤器里也会增加这个字段的属性，可以根据其进行数据过滤，单击功能链接进入该页面即可新建自定义字段、修改已有的自定义字段。

2. 权限用户

1) 经理

经理是整个软件开发过程中较为重要的管理人员。经理在该系统下的使用权限比起管理员来说稍微低一些，由于前面已经详细说明了各个菜单功能的使用，因而这里主要说明经理在本系统所能使用的功能与管理员有什么不同。

对于前面提及的 10 个菜单功能，经理在使用的时候和管理员基本相同，但存在以下几个差异。

(1) 只能对自己的过滤器进行操作，对于管理员设为共有的过滤器只能使用而不能进行操作。

(2) 在“查看 Issue”的时候，可以通过复选框来对某条 Bug 进行命令操作，经理级别的工作人员不能执行“删除”操作，系统会提示“你无权执行该项操作”。

(3) 在“管理”功能中，不能对用户进行管理，包括新建、删除用户等，也不能新建项目，只能管理现有项目的信息。

2) 开发人员

开发人员，就是负责整个软件开发的工作人员。使用 Mantis 这个 Bug 跟踪管理系统，开发人员可以及时地发现和解决软件缺陷。在该系统中，开发人员的权限比起经理来说要稍低一些，从开始登录进入系统就能看出来，其主菜单栏的功能相对少了一些。比起经理和管理员的菜单栏，少了“统计报表”、“管理”这两个功能。

由于前面已经详细说明了各个菜单功能的使用，因而这里主要说明开发人员在本系统所能使用的功能与管理员有什么不同。

开发人员只能设置私有的过滤器，可以共享管理员和经理创建的且属性设置为公有的过滤器。

在“查看 Issue”的时候，可以通过复选框来对某条 Bug 进行命令操作，开发人员不能执行“删除”操作，系统会提示“你无权执行该项操作”。

3) 修改人员

修改人员，就是负责修改 Issue 的工作人员。修改人员的主菜单和开发人员的一样，但其使用权限还是比开发人员稍低一些，下面说明一下操作区别。

不能创建过滤器，但是可以使用由其他工作人员创建且属性被设为公有的过滤器。

在“查看 Issue”的时候，可以通过复选框来对某条 Bug 进行命令操作，修改人员只能进行“复制”、“更新优先级”、“更新状态”、“更新视图状态”操作，对于其他的命令操作则无权执行。

在“我的视图”界面，没有“分派给我的(尚未解决)”状态的 Bug 数据列表，因为对于修改人员来说，不能将 Bug 直接指派给他。因此，相应的界面上没有这类数据的显示。

4) 报告人员

报告人员，就是专门负责提交 Bug 报告的工作人员。报告人员的主菜单也和开发人员、修改人员的一样，但是其使用权限还是比修改人员稍低一些，下面来具体说明一下操作区别。

(1) 不能创建过滤器，但是可以使用由其他工作人员创建且属性被设为公有的过滤器。

(2) 在“查看 Issue”的时候，对 Bug 数据列表不能进行任何命令操作。

(3) 在“我的视图”界面，没有“分派给我的(尚未解决)”状态的 Bug 数据列表，因为对于报告人员来说，也不能将 Bug 直接指派给他。因此，相应的界面上没有这类数据的显示。

5) 查看人员

查看人员，顾名思义就是只具有查看权限的工作人员，在 Mantis 系统中，其权限最低，功能主菜单也相应更少。比起开发人员、修改人员、报告人员的主菜单功能来说，查看人员少了“报告 Issue”的功能，具体的操作区别是：因为查看人员为权限最低的工作人员，因此对于 Mantis 系统的操作功能基本上没有使用权限(除了个人资料的功能外)，而只能查看各个项目的 Bug 流程及具体情况。

6) 分派给我的工作

前面主要是说明各个角色的工作人员在使用该系统的功能时所具备的权限，以下说明工作人员登录该系统后，该如何对分派的工作进行操作。

当工作人员登录系统并进入“我的视图”后，单击“分派给我的(尚未解决)”Bug 数据列表的 Bug 编号链接，则进入分派任务的 Issue 界面。在该界面，Issue 根据各种情况被分成 7 个数据块来显示。在这里主要介绍查看问题详情数据块。

在分派任务的 Issue 界面，上面第一个数据块显示的就是 Issue 的详细资料，在这个数据表格可以进行如下 13 种操作。

(1) 查看注释：单击此链接，可直接跳转到该页面的注释数据。

(2) 发送提醒：单击此链接，可对某个工作人员发送提醒，该提醒会直接在该工作人员的 Issue 监视视图里生成，并在该 Issue 的注释里也增加一条记录。注意：这个功能除了查看人员之外，其他工作人员也可以使用。

(3) Issue 历史：单击此链接，将直接跳转到该页面的 Issue 历史数据。

(4) 打印：单击此链接，将直接在页面上生成这个 Issue 的详细数据，可以通过 IE 提供的打印功能进行打印。

(5) 修改 Issue：如果 Issue 的信息需要修改，则可以单击“编辑”按钮，直接进入 Issue 修改页面。根据实际情况进行修改，然后单击“更新信息”按钮。如果不需要修改了，可以单击“返回 Issue”回到前面的页面。需要注意的是：修改 Issue 的功能只有管理员、经理、开发人员和修改人员能使用。

(6) 分派给：这个功能可以把当前的 Issue 直接指派给选定的工作人员。注意：指派功能只有管理员、经理、开发人员和修改人员能使用，而且只有管理员才能指派给自身。

(7) 将状态改为：这个功能可以对当前的 Issue 流程状态直接进行修改。注意：该功能

只有管理员、经理和开发人员可以使用。

(8) 监视 Issue：这个功能可以把当前这个 Issue 置于所监视的 Issue 范围之内。注意：这个功能除了查看人员之外，其他工作人员也可以使用。

(9) 置顶：这个功能可以将当前这个 Issue 显示在所有“分派给我的(未解决)” Issue 中的第一行。

(10) 创建 Issue 子项：这个功能主要用于创建与当前 Issue 相关的 Issue，单击该按钮进入如图 1-23 所示的页面。页面上用红框标注的就是设定创建 Issue 子项后和当前 Issue 的关系。同时，在“关联”数据块中会增加一条记录。注意：创建子项功能只有管理员、经理、开发人员和修改人员才能使用。

(11) 关闭：这个功能是将该问题关闭，不再修改、讨论。注意：关闭功能只有管理员、经理才能使用。

(12) 移动 Issue：单击此链接可直接将该 Issue 转移到别的项目中去。注意：移动 Issue 功能只有管理员、经理和开发人员才能使用。

(13) 删除 Issue：单击此链接将直接删除该 Issue，只有管理员才有这个权限。

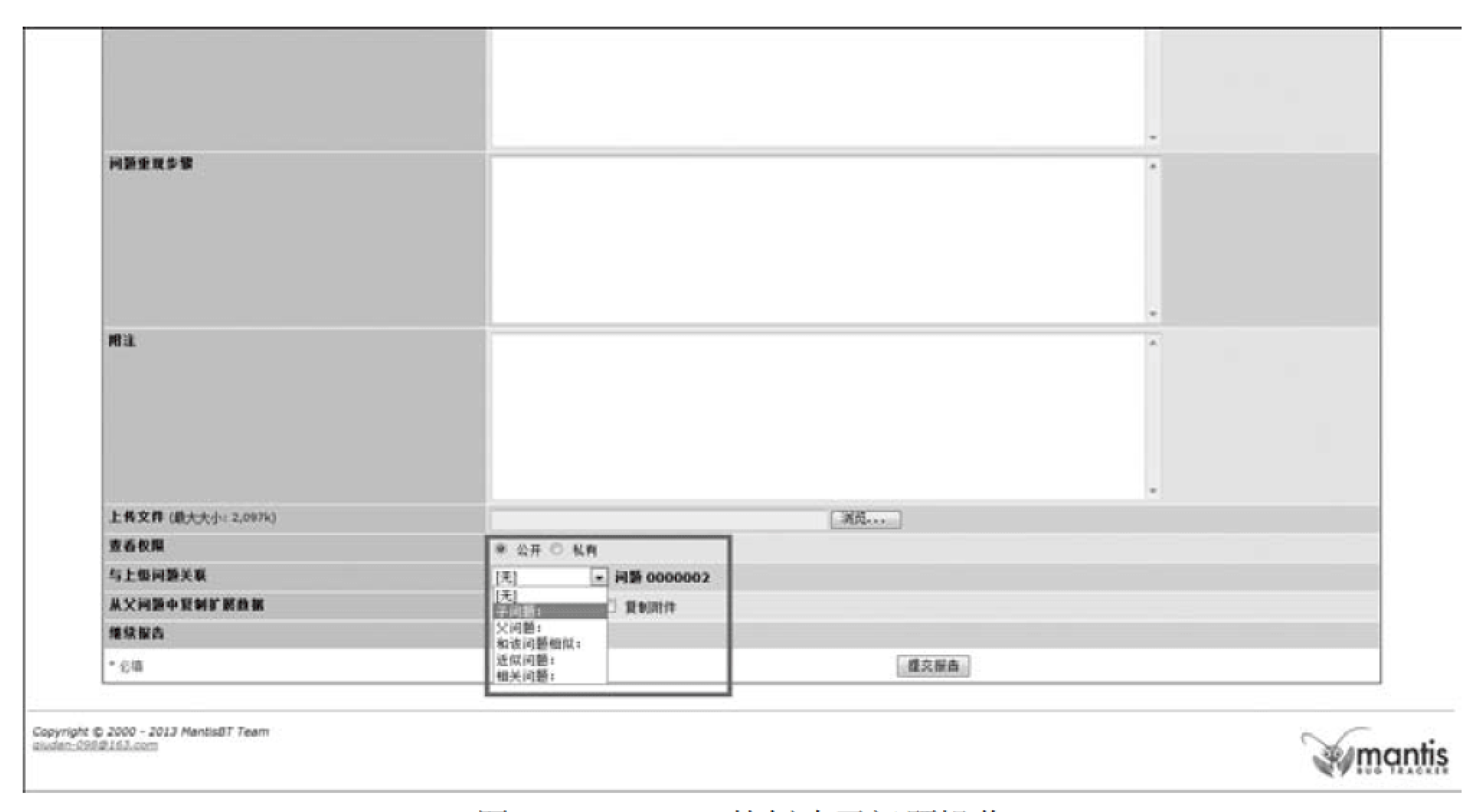


图 1-23　Mantis 的创建子问题操作

1.3　Mantis 应用举例

这里以股票行情分析软件 stock 在 Linux 系统中的升级改造为例，简单介绍一下 Mantis 的应用过程。

1.3.1　Mantis 的应用过程举例

打开 IE，输入“http://localhost/mantis”并按 Enter 键，应该就可以看到 Mantis 的登录页面了。可以用默认用户名 administrator 和密码 root 登录进去，进行管理设置。

Mantis 目录下有一个 admin 目录，如果在 IE 中打开这个目录下的 index.php 文件进行查看，就会知道这个目录是进行 Mantis 管理的，使用这个模块可以检查 Mantis 是否安装完全，对旧版本的 Mantis 进行升级，以及对 Mantis 的页面 CSS 文件进行修改。使用这个管理模块时是不需要用户名和密码的，因此任何人都可以通过这个管理模块来查看 Mantis 的系统信息。而且由于有升级模块，在这里还可以直接对数据库进行修改。因此，如果被未授权的人打开，就可能出现不好的结果。最好按照系统给出的建议，在配置完成后将这个 admin 目录删除。注意一定是删除而不是改名。改名后仍然是可以访问的。在添加一个管理员用户后，删除系统默认的 administrator 用户。

1. 创建项目

进入 Mantis 项目管理页面(如图 1-24 所示)，从菜单中选择“管理”，再选择“项目管理”。



图 1-24　项目管理页面

(1) 添加项目：单击“创建新项目”按钮，本例以股票软件 stock 为被测软件，读者也可以自选项目。在“项目名称”文本框中输入项目名称 stock，其他保持默认即可，如图 1-25 所示。

[用户管理] [项目管理] [标签管理] [自定义字段管理] [平台配置管理] [插件管理] [配置管理]

添加项目

*项目名称	stock
状态	开发中
继承全局类型	
查看权限	公开
描述	Test Stock Software 2012.9.12

添加项目

所有项目 创建新项目

名称	状态	启用	查看权限	描述
stock	开发中	X	公开	Test Stock Software 2013.9.12

图 1-25　项目创建结果

(2) 添加分类：单击新添加的项目 Stock，在分类中“添加分类”，如图 1-26 所示。

2. 提交问题

(1) 选择项目：单击“提交问题”进入如图 1-27 所示页面，选择提交问题所属的项目

（角色为报告员）。



图 1-26　添加分类

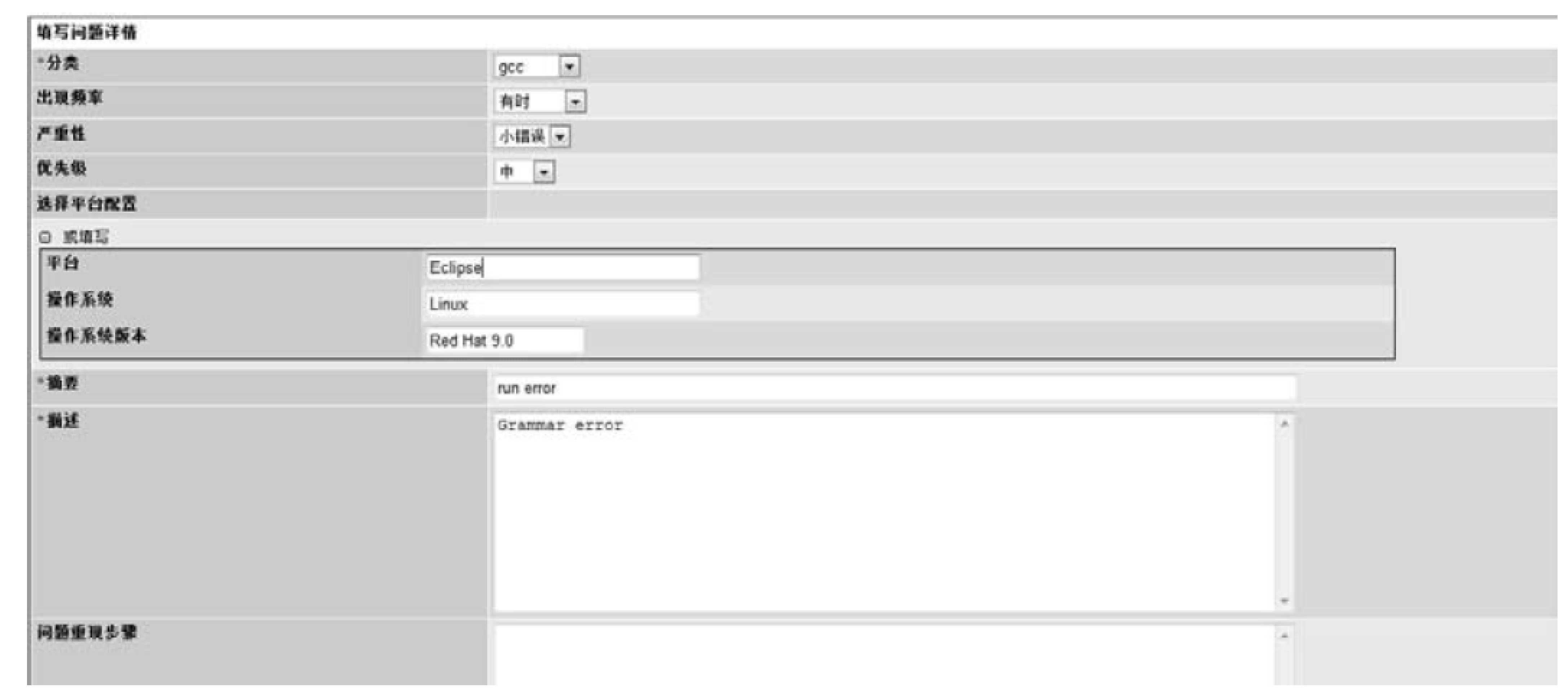


图 1-27　选择项目

(2) 填写项目详细资料：单击“选择项目”按钮，进入提交问题主界面，填写项目的详细资料，完成后单击“提交报告”，如图 1-28 所示。

图 1-28　填写项目详细资料

图中有些选项是打了星的，表示这些是必填内容。填好问题报告后，单击“提交报告”，就可以将此问题提交到系统，系统将通过 E-mail 通知项目组的相关人员。

3. 查看问题

只需单击工具栏上的“查看问题”，就可以看到刚刚创建的问题，如图 1-29 所示。

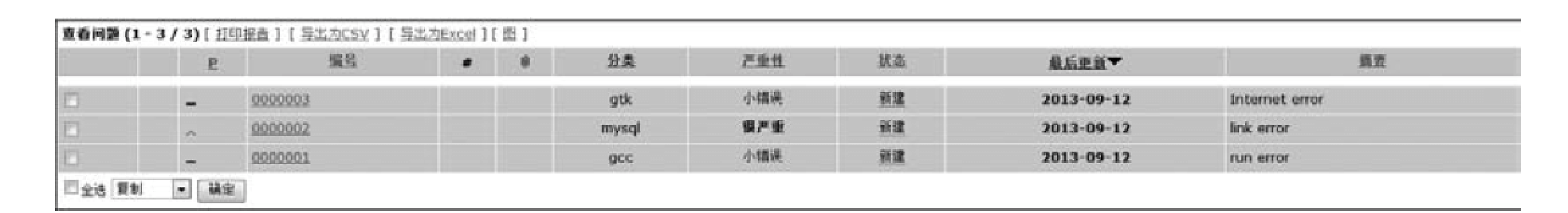


图 1-29　问题查看列表

4. 更新问题

单击问题编号进入问题详情页面，单击“编辑”，对问题进行更新，更新信息后，单击“更新信息”，如图 1-30 所示。问题更新后，会给提交问题的报告员发送一封包含更新信息的邮件。

(1) 查看修改结果，如图 1-31 所示。

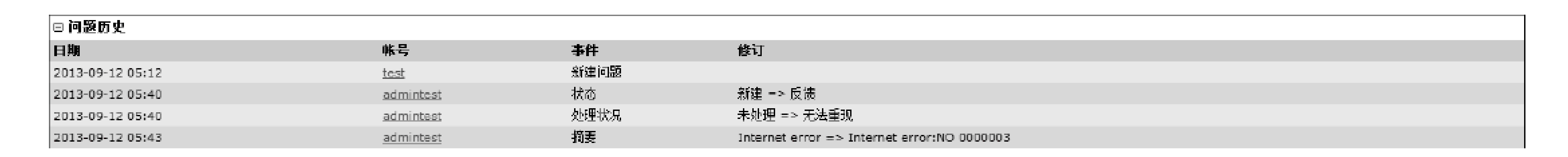
(2) 查看问题更新历史，如图 1-32 所示。



图 1-30　填写问题更新信息



图 1-31　查看修改结果


⊟ 问题历史

日期	帐号	事件	修订
2013-09-12 05:12	test	新建问题	
2013-09-12 05:40	admintest	状态	新建 => 反馈
2013-09-12 05:40	admintest	处理状况	未处理 => 无法重现
2013-09-12 05:43	admintest	摘要	Internet error => Internet error:NO 0000003

图 1-32　查看问题更新历史

5. 创建自定义字段

对创建的字段进行修改，如图 1-33 和图 1-34 所示。

6. 查看最后的缺陷情况

缺陷情况列表如图 1-35 所示。



图 1-33 创建、修改字段并查看结果

[用户管理] [项目管理] [标签管理] [自定义字段管理] [平台配置管理] [插件管理] [配置管理]

修改自定义字段

名称	report
类型	字符串
可能取值	
默认值	
正则表达式	
读权限	复查员
写权限	复查员
最小长度	0
最大长度	0
添加到过滤器	☑
在创建问题时显示	☐
在更新问题时显示	☑
解决问题时显示	☐
关闭问题时显示	☐
报告问题时必需	☐
修改问题时必需	☐
解决问题时必需	☐
关闭问题时必需	☐

修改自定义字段

删除自定义字段

关联自定义字段到项目

关联项目:	stock: [删除] - *report (0)*
所有项目:	Link stock
顺序:	0

关联自定义字段

自定义字段

名称	项目名称	类型	可能取值	默认值
report	1	字符串		

图 1-34 创建、修改字段并查看结果（续）

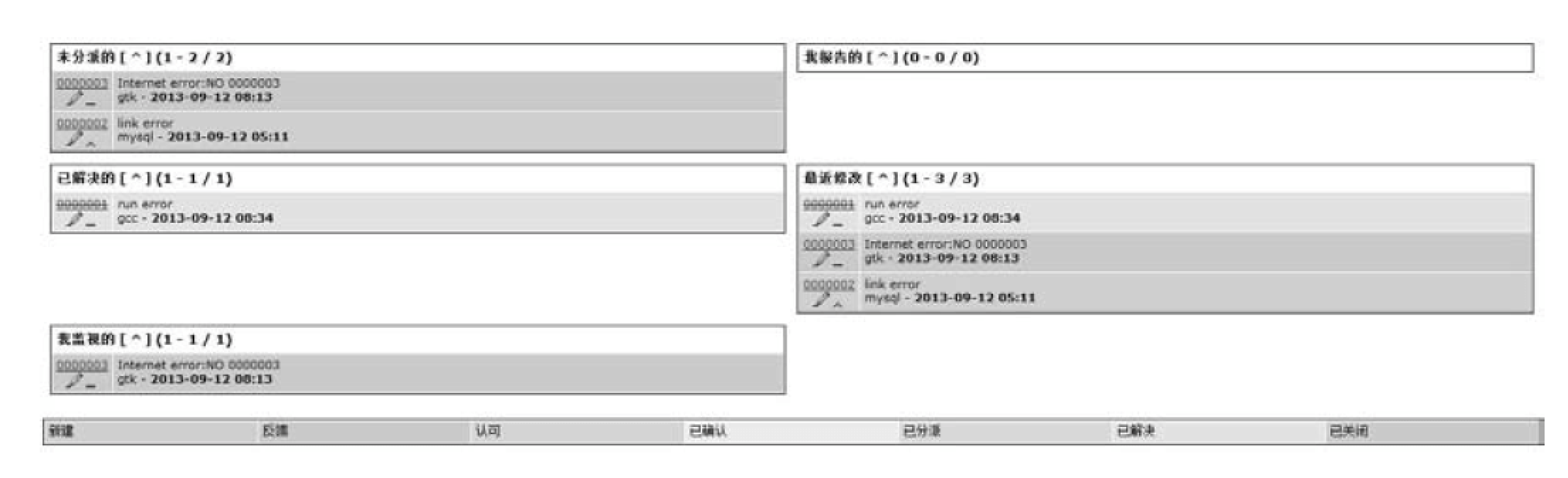


图 1-35　缺陷情况列表

7. 统计报表

单击工具栏上的“报表统计”，以表格的形式对问题进行统计，可“按问题状态”、“按严重性”、“按项目”等进行统计，如图 1-36 所示。



图 1-36　缺陷统计

1.3.2　stock 软件中的缺陷处理流程举例

(1) 管理员创建项目之后，项目经理 admin 对测试项目(stock)进行编辑，如图 1-37 所示。

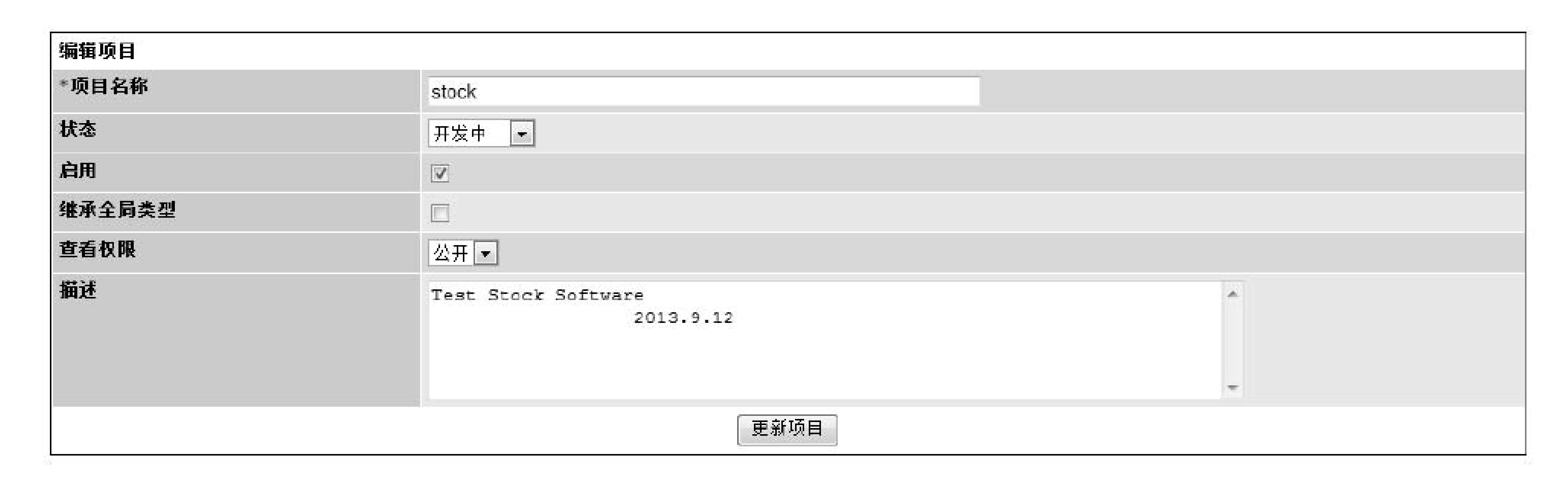


图 1-37　编辑测试项目 stock

(2) 添加分类，还可以设置、修改版本信息，如图 1-38 所示。



图 1-38　添加测试分类及版本信息

(3) 测试人员 kerry（报告人员）发现问题(stock 软件安装编译的时候发生问题，软件终止且不能继续运行)，编写缺陷报告后提交：stock 软件出现缺陷。缺陷状态自动设置为“新建”，如图 1-39 所示。



图 1-39　编写问题报告

(4) 开发人员 amyny 登录后在查看问题页面时看到状态为“新建”的 Bug 后，打开问题报告详细页面，按照问题重现步骤实现 Bug，发现 Bug 可以重现，将缺陷状态改为“已确认”，如图 1-40 所示。



图 1-40　将缺陷状态设置为“已确认”

(5) 项目经理审查后，表示对该 Bug 认可，将缺陷状态设置为“认可”，并将其分派

给开发人员 amyny，如图 1-41 所示。



图 1-41　将缺陷状态设置为“认可”并分派

(6) amyny 发现分派给自己的问题，将问题解决后更新缺陷报告(说明缺陷已经被解决，并说明软件的现状)，并更新缺陷状态为“已解决”，如图 1-42 所示。



图 1-42　将缺陷状态设置为“已解决”

(7) kerry 发现 Bug 已经被修复，对该 Bug 进行验证，若验证未通过，可以重启问题，若通过验证，不进行任何操作，如图 1-43 所示。



图 1-43　对缺陷进行验证

(8) 项目经理发现问题被解决，且未被重启，将该问题关闭，如图 1-44 所示。

关闭问题
分派给 amyny
修正版本 V2.0
添加注释 问题解决，关闭问题
查看权限 私有
关闭问题

图 1-44 关闭问题

(9) 现在任何级别的用户查看问题时，都将发现该问题已经不存在了，如图 1-45 所示。



查看问题 (1 - 3 / 3) [打印报告] [导出为CSV] [导出为Excel] [图]

		P	编号	#		分类	严重性	状态	最后更新	摘要
		^	0000002	2		[stock] mysql	很严重	已分派 (admintest)	2013-09-14	link error
		–	0000001			[stock] gcc	小错误	已确认 (admintest)	2013-09-14	run error
		–	0000004			[stock] gcc	小错误	已解决 (admintest)	2013-09-14	compiling error

全选 移动 确定

新建 反馈 认可 已确认 已分派 已解决 已关闭

图 1-45 “查看问题”页面中已经没有该问题了

实 验 习 题

1. 建立 Bugzilla 缺陷管理环境，并通过一个具体的实例，详细说明 Bugzilla 缺陷管理的功能和流程。

2. 选择 Bugfree、Bugtrack 或其他缺陷管理工具，并将它们与 Mantis 进行比较，评判它们对缺陷管理流程的支持及各自的特点。

第2章　软件测试管理

软件测试最重要的是有效的测试管理。测试管理包括对人的管理、对流程的管理、对具体版本的管理等。测试要考虑的所有问题都可以列入此范畴，例如，测试人员的分工、测试规程的制定、测试流程的裁减、采用什么测试流程、测试设计怎样操作、测试执行如何计划、测试度量怎样进行等。

软件测试管理实际上是一系列活动，可以对各阶段的测试计划、测试用例、测试流程、测试文档等进行跟踪、管理并记录其结果，以实现测试的有效控制和管理，进一步提高测试的效率和质量。

作为软件测试管理的实践活动内容，测试管理中最重要的，也是学生必须掌握的是测试执行与缺陷跟踪阶段的管理活动和管理手段，而测试执行与缺陷跟踪阶段的管理重点是保证测试能按照计划顺利进行和有效实施。通过规范测试流程，加强测试的有效性检查，及时报告测试进度，促进测试团队的交流，成为决定这一阶段工作成败的关键。

2.1　软件测试管理工具

对于测试管理来说，有许多令人生畏而且不可避免的挑战，好消息是有大量管理工具可以应对这些挑战。使用测试管理工具对软件的整个测试输入、执行过程和测试结果进行管理，可以提高测试的效率、测试时间、测试质量、用例复用、需求覆盖等。

由于软件项目测试越来越复杂，单纯增加测试人手已不能解决问题，需要一个软件工具来管理测试。软件测试工具种类繁多，主流的测试工具可以分为测试管理工具、负载测试工具、功能测试工具等。相比于测试管理工具，其他的具体的测试技术工具都是次要因素，比如各种测试工具、“白盒”测试、“黑盒”测试等。这些具体的测试技术都是相对比较“固化”的，测试技术人员可以慢慢地掌握它们。但在评价一个测试团队的测试水平，或者在评估一个项目的测试质量时，不会因为这些具体的测试技术掌握与否而断言，更多的考虑应是它的测试管理过程和软件测试管理工具的运用。

2.1.1　软件测试管理工具应具备的功能

一个完整的软件测试管理工具，应能用于测试的计划、文档和缺陷跟踪等各种测试行

为的管理，并能提供对人工测试和自动测试基于过程的分析、设计和管理功能，把应用程序测试中所涉及的全部任务集成起来；包括测试中包含的所有工作，跟踪测试资产中的依赖关系和相互关联，并能对质量目标进行定义、测量和跟踪。

一般而言，软件测试管理工具应具备如下功能。

1. 测试计划和进度管理

软件测试管理工具应能跟踪项目测试工作是否按照预期计划和进度开展，对软件产品从功能分类、相关人员分配、测试用例编写、测试用例执行到问题报告生成的整个测试流程进行系统、有效地管理。项目管理人员可以通过该工具随时了解、监控被测项目的执行进度和软件问题的处理状态，为测试人员和开发人员的工作提供有效的考核依据，保障软件开发和测试的顺利进行。

2. 测试资产管理

测试资产管理包括测试计划、测试用例、测试脚本、测试报告的创建与维护以及缺陷跟踪，保证测试资产之间是可跟踪的、一致的。无论是开发人员、测试人员还是项目管理人员，都可以随时编写、修改和查阅测试用例和软件问题报告，能对测试用例和问题报告进行长期保存以避免流失。对测试用例的编写与执行情况进行全程记录，便于追踪测试用例在各个测试阶段的执行过程，及时调整测试策略与方法。最后，还应提供统一的软件问题报告模板与测试用例模板，使测试人员能够更加准确、详细地编写测试用例和描述软件问题，保证测试用例与软件问题报告描述的一致性，便于积累、分类与查询。

3. 测试质量评估与报表

软件测试管理工具应具有实用的统计功能，可以从各种角度建立分析统计报表，以便及时掌握测试执行情况，例如，软件问题的有效发现率、有效修复率及各项测试工作的进度。好的软件测试管理工具必须提供所有相关信息的完整和正确的报告，所有这些测试报告要指导如何对测试工作的不同结果进行分析和沟通，并为不同的项目角色能随项目的进展对变化做出合适反应提供决策依据，以及帮助判断测试的当前状态和质量水平。

2.1.2　软件测试管理工具的选择

面对市场上众多的商业化软件测试管理工具，选择就成了一个比较重要的问题。在选用软件测试管理工具的时候，建议从以下几个方面来权衡和考虑。

1. 功能考量

功能当然是人们最关注的内容，选择一个测试工具首先就是看它提供的功能。当然，这并不是说测试工具提供的功能越多就越好。在实际的选择过程中，适用才是根本。“钱要花在刀刃上”，为不需要的功能花费金钱实在不是明智的行为。事实上，目前市面上同类的软件测试工具之间的基本功能都是大同小异，各种软件所提供的功能也大致相同，只

不过侧重点不同。例如报表功能，查看测试管理工具所生成报告的人员不一定对测试很熟悉；因此，测试工具能否生成合乎要求的结果报表，能够以什么形式提供报表是其中一个考察因素。

2. 测试管理工具的集成能力

测试管理工具的集成能力也是必须考虑的因素，这里的集成包括两个方面的意思：首先，测试管理工具能否和开发工具进行良好的集成；其次，测试管理工具能否和其他测试工具进行良好的集成。另外，与操作系统和开发工具的兼容性也需要考虑。测试管理工具可否跨平台，是否适用于公司目前使用的开发工具，这些也都是在选择一个测试管理工具时必须考虑的问题。

3. 能否与现有的测试管理保持连续性和一致性

引入测试管理工具的目的是使测试管理自动化，引入工具时需要考虑工具引入的连续性和一致性。也就是说，对测试工具的选择必须有一个全盘的考虑，要考虑到公司的实际情况，避免盲目引入测试管理工具。因为并不是每种测试工具都适合公司目前的实际情况，如果怀着美好的愿望花了不小的代价引入测试工具，一年半载后测试工具却成了摆设，就不仅是浪费了资金，而且还会对软件项目的开发进度产生严重影响。

4. 是否具备测试团队管理功能

测试管理工具应具有测试团队管理功能，要能根据人员的分工和职能不同来严格划分权限，从而明确测试任务，并保证测试质量。如测试人员的绩效考核、人员的技术定位、人员激励等。

测试管理工具应能有效处理人力资源不足问题或者能更充分地利用人员，能协调运用任何人力资源，而不管它们在什么地方。这些重要的人力资源可能存在于不同的地方，这需要仔细有效的协同合作，才能使大多数测试人员和其他人员参与到测试管理中，对于那些跨越一个或更多场所或者测试团队的项目来说，还包括区域场所和团队协调。

2.1.3　常用软件测试管理工具介绍

在软件测试管理过程中，免不了要借助一些软件测试管理工具，以对测试进行管理。一般而言，软件测试管理工具用于对测试计划、测试用例和测试实施进行管理，另外还包括对缺陷的跟踪管理。具有代表性的商用软件测试管理工具有 IBM Rational 公司的 Test Manager、Compureware 公司的 TrackRecord、HP 公司的 ALM 等，深受中小企业欢迎的开源软件测试管理工具有 TestRunner(更名为 Testopia)和 TestLink，国产的软件测试管理工具有上海泽众软件科技有限公司的TestCenter。

1. TestManager

TestManager 的优点是有自己的客户端、响应速度较快、Case 管理功能强大(可以任意

地组合自己的 Test Case；可以区分不同 build 保存的测试结果、集成了强大的数据生成器(dataPool)、与第三方报表完全集成、可生成多种且完整的数据报表及图表、可与 IBM 的其他组件(如 CQ、Robot、Requisite Pro 等)无缝结合，总之是很实现完整的测试管理及测试执行平台。

其不足之处是：没有权限管理功能；支持的数据库类型太少，尤其是不支持 Access 数据库，不能支持太多的并发访问用户数；不支持 Test Case 的实例化功能；不能预先安排 Case 的执行任务给相应的测试人员；有用户反映有莫名其妙的错误，找不到解决方案。

2. ALM

ALM 的优点是有些功能比 TestManager 更强大，如可以进行需求管理和缺陷管理；有测试用例执行跟踪的功能，可以统计测试用例对需求及缺陷的覆盖；可以和 VSS 集合，对测试用例进行版本控制；有灵活的缺陷定制，可以和 ClearQuest 紧密结合，使两个缺陷工具的数据得到同步；采用 Web 形式的界面，界面较友好。

其不足之处是：每个项目库对同时在线人数有限制；可能存在部分不稳定性，但是基本功能没有问题；对 Close 的测试集没有状态标记；快速执行测试用例的时候，不能看到测试用例内容，而快速执行是常用到的功能；不能实现自动多级连动，需要硬编码。

3. TestRunner

这是与 Bugzilla 集成的一款测试管理工具，使用时有一定的局限性，但其易用性的功能，特别是与 Bugzilla 共同进行的管理效果显著，因而受到了许多测试工程师的青睐，其优点是：开源免费；采用 Web 方式的管理界面；自动邮件提醒；和缺陷管理系统 Bugzilla 结合紧密，有测试用例执行管理；测试用例可以分优先级；测试用例可以有评审功能。(测试用例有不同的状态)

不足之处是：安装设置较烦琐，测试用例必须按照一个步骤对应一个验证点的形式来编写。

4. TestLink

这是一款优秀的开源测试管理软件。与 TestRunner 不同，TestLink 可以和更多的缺陷管理软件进行集成(Bugzilla、Mantis 等)，因此，如果选用了其他的缺陷管理软件，则推荐使用 TestLink 进行测试。TestLink 用于测试过程中的管理，通过使用 TestLink 提供的功能，可以将测试过程从测试需求、测试设计到测试执行完整的流程管理起来。同时，它还提供了测试结果的多种统计和分析图表，使开始测试工作和测试结果分析工作变得更简单。TestLink 是 SourceForge 的开源项目之一。

5. TestCenter

TestCenter是由上海泽众软件科技有限公司研发的一款功能强大的测试管理工具，它可以帮助用户实现测试用例的过程管理，对测试需求过程、测试用例设计过程、业务组件设计实现过程等整个测试过程进行管理。通过实现测试用例的标准化(即每个测试人员都能够

理解并使用标准化后的测试用例)，降低了测试用例对个人的依赖；提供测试用例复用，使测试用例和测试脚本能够被复用，以保护测试人员的资产；提供可伸缩的测试执行框架，提供自动测试支持；提供测试数据管理，帮助用户统一管理测试数据，降低测试数据和测试脚本之间的耦合度。

(1) 测试需求管理：支持测试需求树，树的每个节点是一个具体的需求，也可以定义子节点作为子需求。每个需求节点都可以对应到一个或多个测试用例。

(2) 测试用例管理：测试用例允许建立测试主题，通过测试主题来设置测试用例的使用范围，实现有效的测试。

(3) 测试业务组件管理：支持测试用例与业务组件之间的关系管理，通过测试业务组件和数据“搭建”测试用例，实现了测试用例的高度可配置和可维护性。

(4) 测试计划管理：支持测试计划管理、测试计划多次执行(执行历史查看)，测试需求范围定义、测试集定义。

(5) 测试执行：支持测试自动执行(通过调用测试工具)；支持在测试出错的情况下执行错误处理脚本，保证出错后的测试用例脚本能够继续被执行。

(6) 测试结果日志查看：提供截取屏幕的日志查看功能。

(7) 测试结果分析：支持多种统计图表，如需求覆盖率图、测试用例完成的比例分析图、业务组件覆盖比例图等。

(8) 缺陷管理：支持从测试错误到曲线的自动添加与手工添加；支持自定义错误状态、自定义工作流的缺陷管理过程。

2.2　软件测试管理工具 TestLink 应用

作为基于 Web 的测试管理系统，TestLink 的主要功能包括：测试需求管理、测试用例管理、测试用例对测试需求的覆盖管理、测试计划的制定、测试用例的执行以及大量测试数据的度量和统计。TestLink 能够加速并简化测试的这个管理流程，动态地收集、组织测试用例，跟踪与整合相关联的测试，获取并报告详细的信息，帮助开发人员管理这个测试流程，并且还能够帮助用户自定义以适用项目需求与过程。

2.2.1　TestLink 功能介绍

TestLink 针对整个测试过程进行管理，包括创建测试脚本、执行测试、跟踪测试结果等。除此之外，它还能够让测试、测试开发和测试报告变得更加简单。其功能主要表现在：动态地收集和组织测试用例，跟踪测试执行后的测试结果，跟踪独立测试的准确信息，获取并详细地报告测试结果，可以帮助测试人员更好地管理整个测试过程等。

TestLink 在以上功能的表现上自然有它自身的特点，主要表现在：Web 方式访问；测试计划中的每个产品的测试都遵循测试流程；用户可以自定义角色；关键字被用于支持深

层次的测试组织；测试可以根据优先级指派给测试员，定义里程碑；提供测试报告，支持将文档以 HTML、Word 或 Excel 格式导出，可以直接通过这个工具将测试报告以邮件形式发送出去；支持本地化和国际化；可结合通用的 Bug 跟踪系统，如 Bugzilla、Mantis 和 Jira；基于测试的需求管理。

2.2.2　TestLink 应用环境建立

TestLink 是一个开源软件，采用 PHP 开发。因此，如果想使用 TestLink，则至少需要安装 PHP，否则无法使用。另外，TestLink 是一个测试管理平台，是一个 B/S 模式的系统，因此在搭建 TestLink 的过程中，还需要有一个服务器来放置 TestLink 的服务。可以选用的有 IIS，也可以使用 Apache HTTP Server。最后，由于测试管理和数据息息相关，需要选用一个数据库来存放测试数据、用户信息、项目信息等，在这里选用 MySQL 数据库。

关于 Apache+MySQL+PHP 的安装方法，在前面介绍安装 Mantis 工具的时候就已介绍过，现在不再详细说明。

下面介绍 TestLink 的安装。

1. 解压 TestLink 源文件

从 http://sourceforge.net/projects/testlink 上下载 Testlink 的最新版本，目前的最新版本是 1.9.8，将其解压缩，重命名为 testlink，复制到 E:\xampp\htdocs 目录下。

2. 安装 TestLink

完成上述步骤后，打开浏览器，在地址栏中输入“http://localhost/testlink”进入安装页面，如图 2-1 所示。

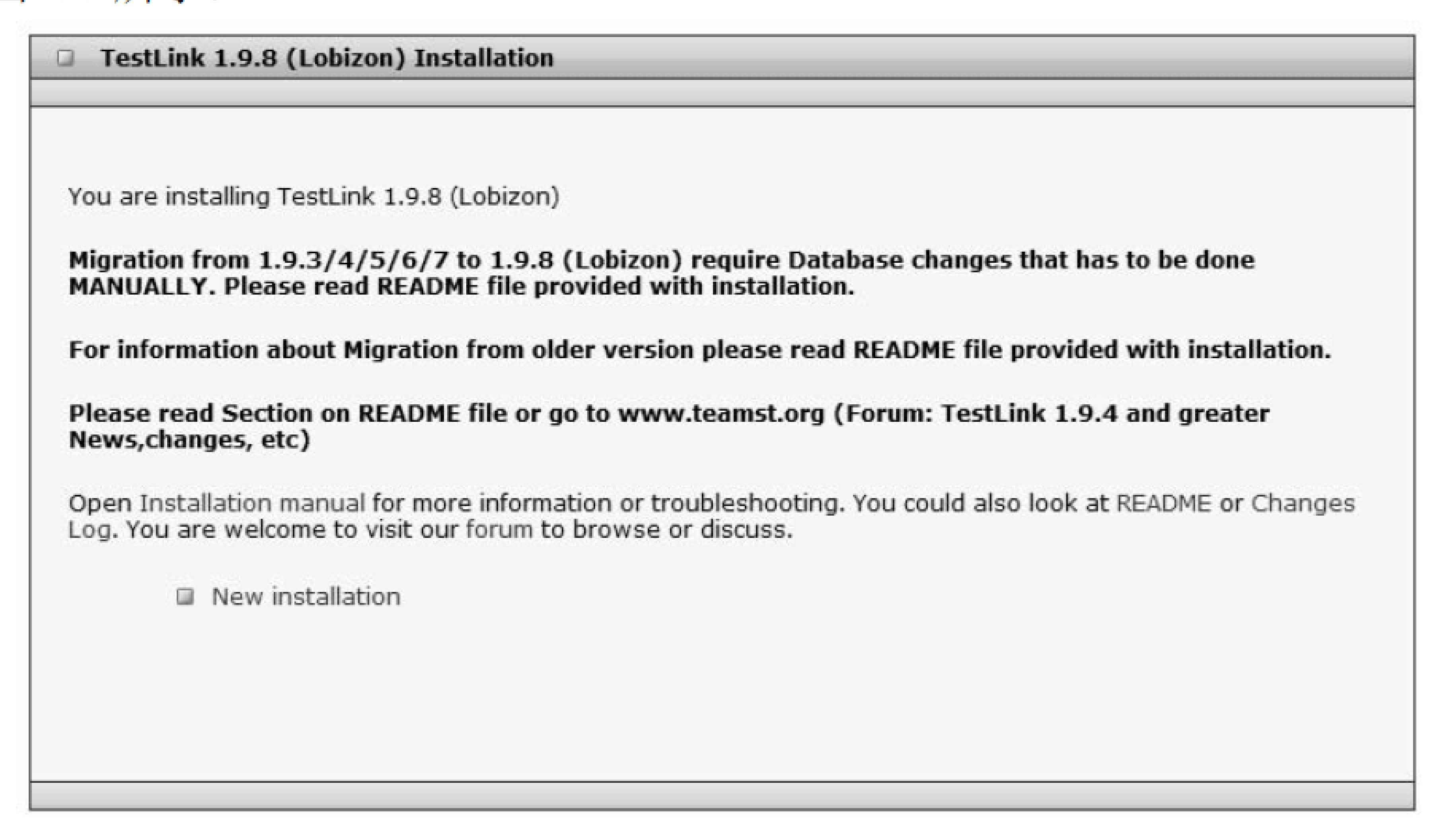


图 2-1　TestLink 安装页面

选择 New installation 复选框，进入安装页面。TestLink 在检查系统配置时，会出现报错（这个问题只会出现在 1.9.4 及之后的版本），如图 2-2 所示。

Read/write permissions

For security reasons we suggest that directories tagged with [S] on following messages, will be made UNREACHEABLE from browser.
Give a look to README file, section 'Installation & SECURITY' to understand how to change the defaults.

Checking if E:\xampp\htdocs\testlink\gui\templates_c directory exists	OK
Checking if E:\xampp\htdocs\testlink\gui\templates_c directory is writable (by user used to run webserver process)	OK
Checking if /var/testlink/logs/ directory exists [S]	Failed!
Checking if /var/testlink/upload_area/ directory exists [S]	Failed!

图 2-2　系统检测报错

解决方法：

(1) 打开 testlink 目录下的 config.inc.php 文件

找到$tlCfg->log_path = '/var/testlink/logs/'; /* unix example */

注释掉该句，添加如下内容：

```
$tlCfg->log_path = 'testlinkDir/logs/';
```

(2) 找到$g_repositoryPath = '/var/testlink/upload_area/';　/* unix example*/

注释掉该句，添加如下内容：

```
$g_repositoryPath = 'testlinkDir/upload_area/';
```

注意：testlinkDir 表示安装目录路径，本次安装为 E:/xampp/htdocs/testlink。

做出上述修改后，当再次检验时，成功通过，如图 2-3 所示。

Read/write permissions

For security reasons we suggest that directories tagged with [S] on following messages, will be made UNREACHEABLE from browser.
Give a look to README file, section 'Installation & SECURITY' to understand how to change the defaults.

Checking if E:\xampp\htdocs\testlink\gui\templates_c directory exists	OK
Checking if E:\xampp\htdocs\testlink\gui\templates_c directory is writable (by user used to run webserver process)	OK
Checking if E:/xampp/htdocs/testlink/logs/ directory exists [S]	OK
Checking if E:/xampp/htdocs/testlink/logs/ directory is writable (by user used to run webserver process)	OK
Checking if E:/xampp/htdocs/testlink/upload_area/ directory exists [S]	OK
Checking if E:/xampp/htdocs/testlink/upload_area/ directory is writable (by user used to run webserver process)	OK

图 2-3　系统检测通过

单击 continue，进入数据库配置页面，安装页面中原有的数据可以保持默认，在其余的填写项中输入数据库的用户名和密码(也就是 MySQL 数据库的用户名和密码)，如图 2-4 所示。单击“下一步”按钮即可开始安装 TestLink。

最后，转入安装成功页面，如图 2-5 所示。

3. 查看安装是否成功

访问 http://localhost/testlink/index.php，如果能显示如图 2-6 所示的登录界面，那么就说明安装成功了。

Table prefix (optional)

Note: This parameter should be empty for the most of cases.
Using a Database shared with other applications: *Testlink can be installed (using this installer) on a existing database used by another application, using a table prefix.*
Warning! PART OF INSTALLATION PROCESS CONSISTS on dropping all TestLink tables present on the database/schema (if any TestLink table exists). Backup your Database Before installing and load after this process.

Set an existing database user with administrative rights (root):

Database admin login root
Database admin password ••••

This user requires permission to create databases and users on the Database Server.
These values are used only for this installation procedures, and is not saved.

Define database User for Testlink access:

TestLink DB login root
TestLink DB password ••••

This user will have permission only to work on TestLink database and will be stored in TestLink configuration.
All TestLink requests to the Database will be done with this user.

After successfull installation You will have the following login for TestLink Administrator:
login name: admin
password : admin

图 2-4　数据库文件配置

TestLink 1.9.8 (Lobizon)　　**TestLink 1.9.8 (Lobizon) - New installation**

TestLink setup will now attempt to setup the database:

Creating connection to Database Server:OK!

Database testlink does not exist.
Will attempt to create:
Creating database `testlink` :OK!
Creating Testlink DB user `root` :OK! (ok - user_exists ok - grant assignment)
Processing:sql/mysql/testlink_create_tables.sql
OK!
Writing configuration file:OK!

YOUR ATTENTION PLEASE:
To have a fully functional installation You need to configure mail server settings, following this steps

- copy from config.inc.php, [SMTP] Section into custom_config.inc.php.
- complete correct data regarding email addresses and mail server.

Installation was successful!
You can now log in to Testlink (using login name:admin / password:admin - Please Click Me!).

图 2-5　安装成功页面



图 2-6　启动 TestLink

2.2.3　TestLink 使用流程

基于 TestLink 的测试管理流程一般包括：创建项目、创建测试需求、创建测试计划、创建测试用例、为需求指派用例、为计划添加用例、分配测试任务、执行测试/报告 Bug 并跟踪、查看分析结果。详细流程如图 2-7 和图 2-8 所示。

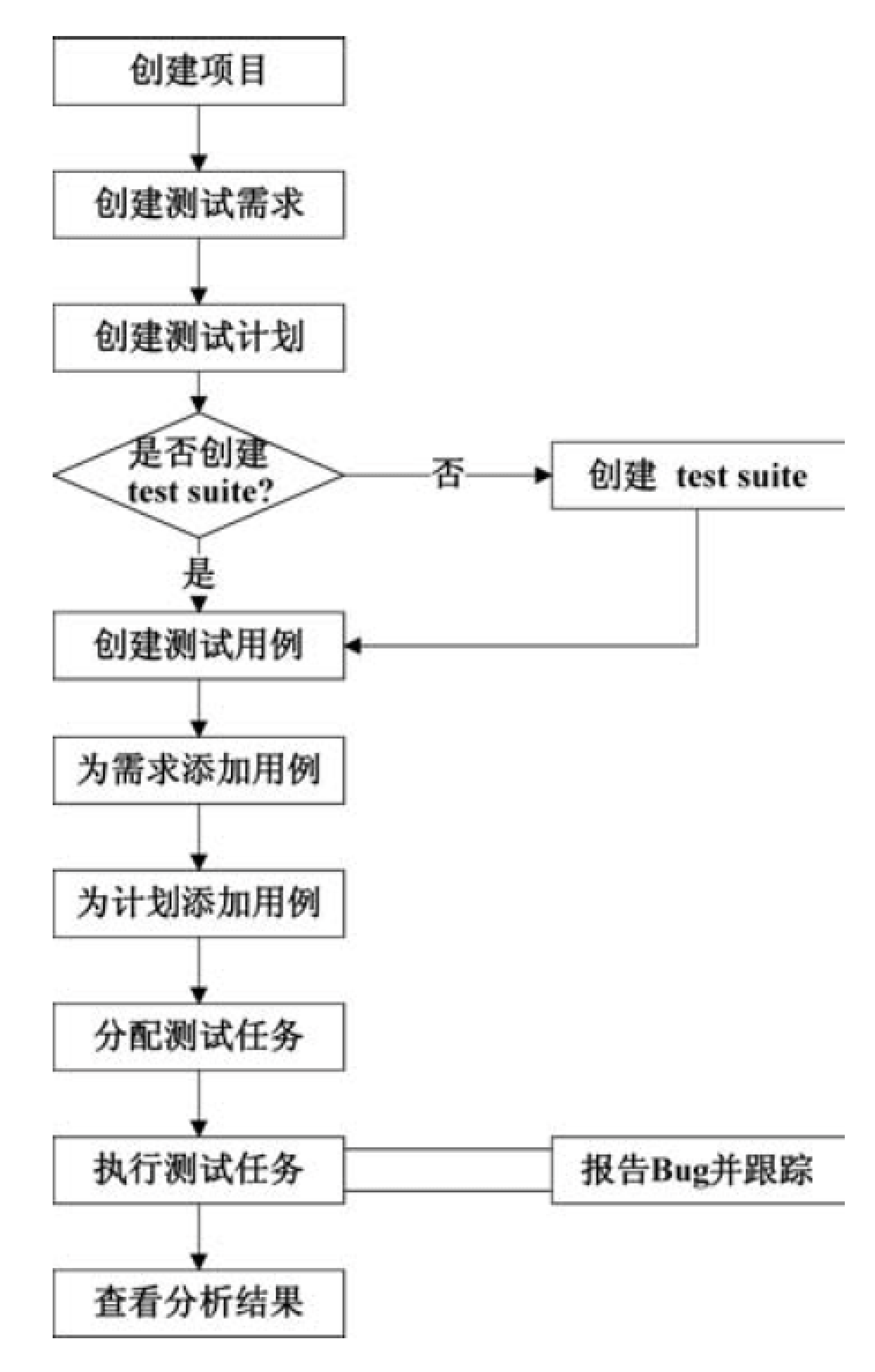


图 2-7　基于 TestLink 的测试管理流程

其中，在创建测试用例之前，需要确定测试用例所隶属的 test project 和 test suite 都已经存在。因为 TestLink 用例数的层次为 test project - test suite - test case，而 test project 在一开始就已经创建，所以现在需要判定 test suite 是否已经创建。

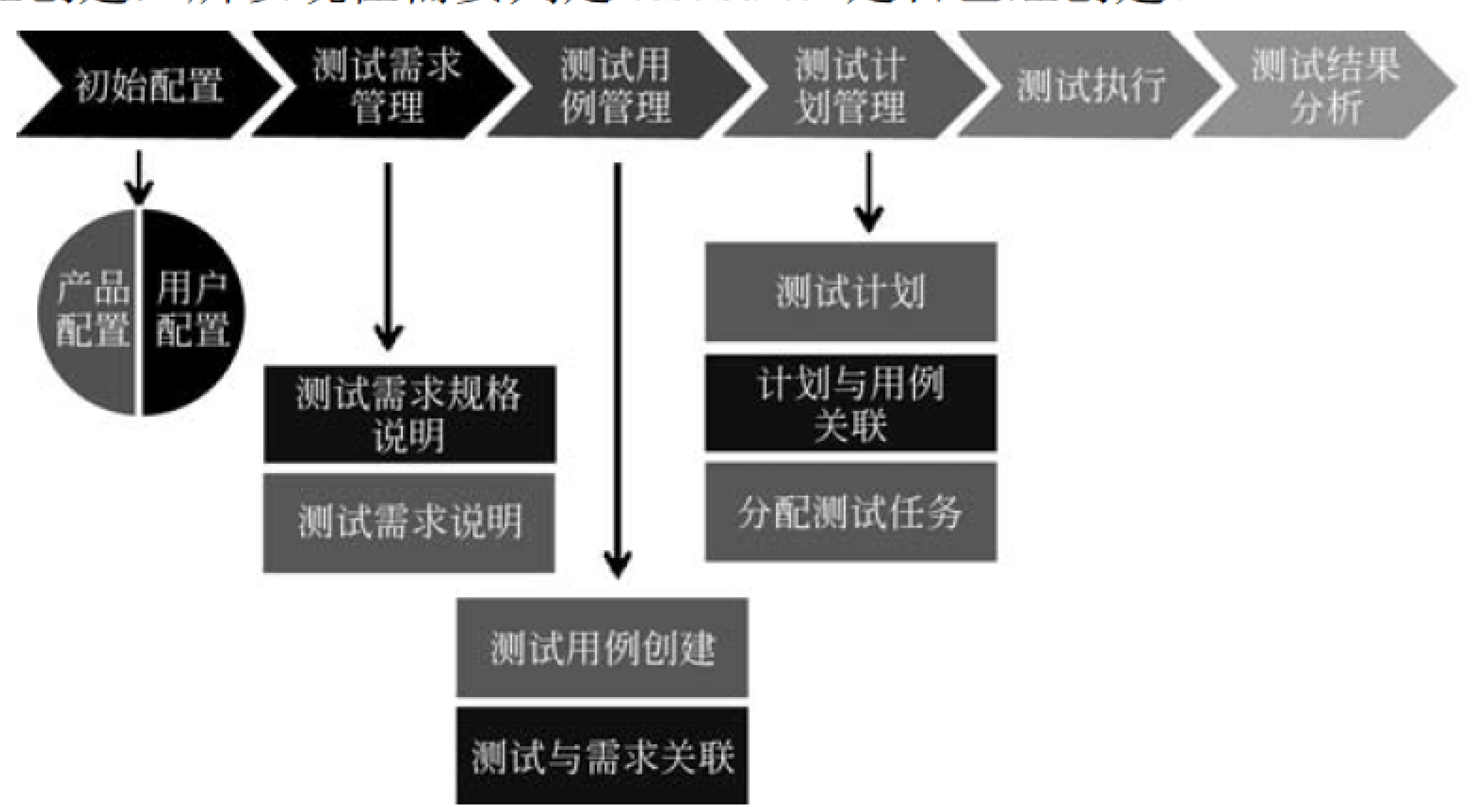


图 2-8　TestLink 的工作流程详解图

2.2.4　TestLink 应用举例

下面将从 TestLink 初始配置、确定测试需求、测试用例管理、制定测试计划、测试报告与度量以及与缺陷管理系统的集成等方面对 TestLink 的具体应用进行介绍。

1. 初始配置(设置邮箱、用户、产品)

1) 邮箱设置

打开 E:\xampp\htdocs\testlink 目录下的 config_inc.php 文件，找到以下语句并进行修改（以 163 邮箱为例）：

```
$g_smtp_host        = 'smtp.163.com';          #SMTP 服务器
$g_tl_admin_email     = 'xxx@163.com';         #管理员邮箱
$g_from_email        = 'xxx@163.com';          #发送邮箱
$g_return_path_email  = 'xxx@163.com';
$g_mail_priority = 5;                          #邮件级别设定 1 为紧急、5 为不紧急、0 为空
$g_phpMailer_method =PHPMAILER_METHOD_SMTP;
$g_smtp_username     = 'xxx';                  #只在 SMTP 服务器需要用户名密码验证时填写
$g_smtp_password     = 'yyy';                  #只在 SMTP 服务器需要用户名密码验证时填写
$g_smtp_connection_mode = '';                  #默认为空，可以设置 SSL、TLS
$g_smtp_port = 25;
```

2) 用户设置

打开浏览器，在地址栏输入“http://localhost/testlink”。

系统为 TestLink 创建了一个默认的管理员账号，用户名/密码为 admin/admin。可以使用这个账号访问 TestLink，如果是第一次访问，访问后 TestLink 会要求用户创建一个新的测试项目，如图 2-9 所示。

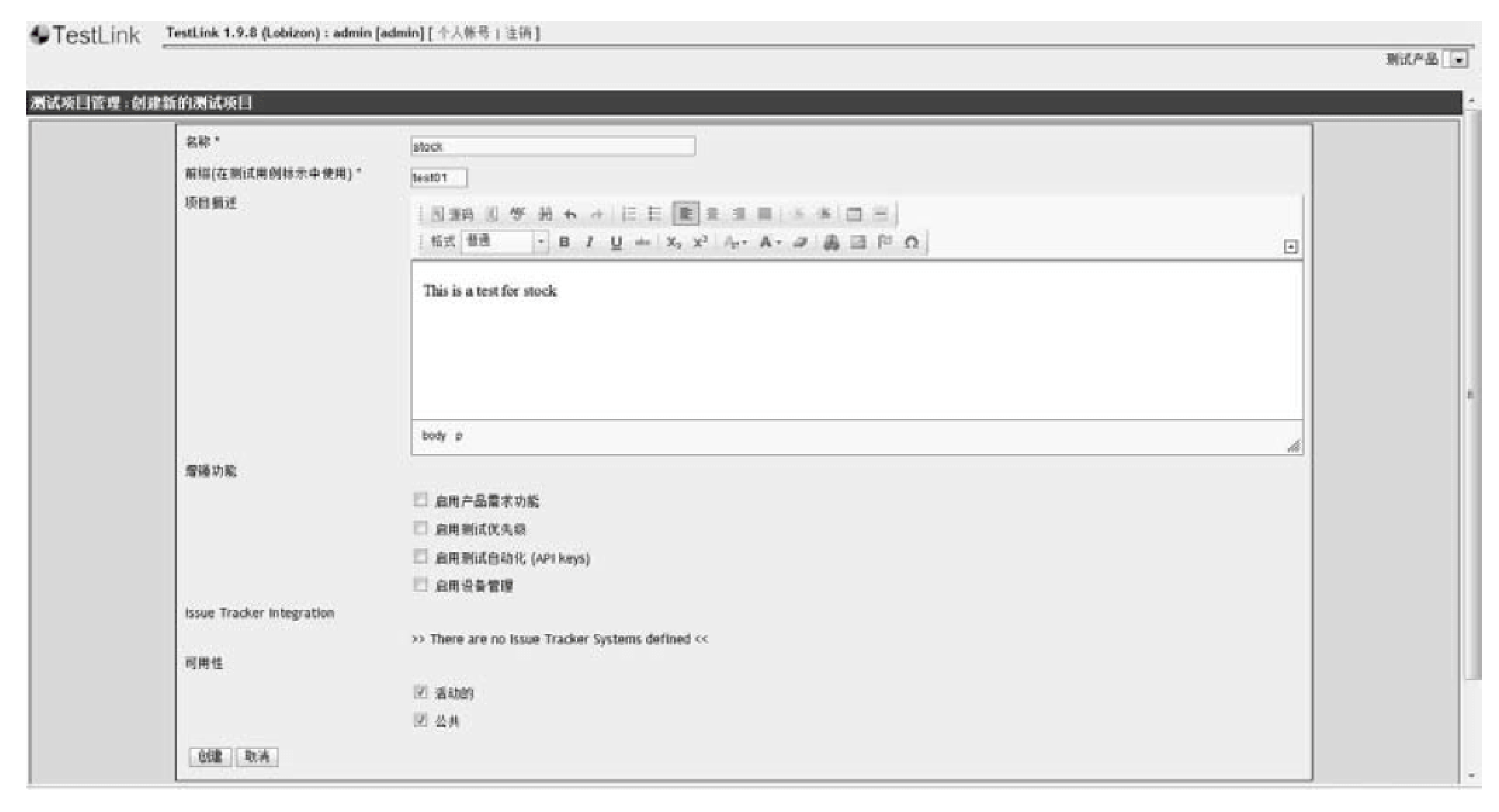


图 2-9　创建新的测试项目

只有在创建了测试项目之后，页面上才会出现功能栏，如图 2-10 所示。

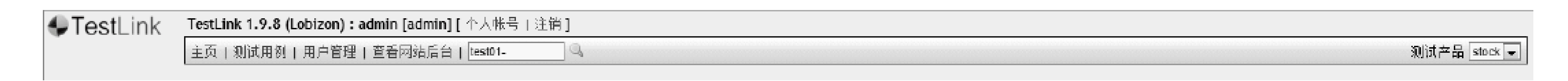


图 2-10　TestLink 的功能栏

在 TestLink 系统中，每个用户都可以维护自己的私有信息。admin 可以创建用户，但无法看到其他用户的密码。在用户信息中，需要提供 E-mail 地址，如图 2-11 所示。这样当用户忘记密码时，便可以通过 E-mail 来获得。

TestLink 有 6 种不同的默认权限级别，对于管理员来说，通过用户管理链接可以很容易地改变权限。这些权限如下所示。

(1) Guest：只能查看测试用例和项目度量。

(2) Tester：只能执行分配给他们的测试用例。

(3) Test Designer：可以开展测试用例和测试需求的所有工作。

(4) Senior Tester：可以查看、创建、编辑和删除测试用例，并且可以执行测试用例，但是不能管理测试计划、管理产品、创建里程碑或分配权限。(针对初级测试员和高级测试员。)

(5) Leader：拥有一个 Tester 所有的权限，并且可以管理测试计划、分配权限、创建里程碑和管理关键字。

(6) Admin(Administrator)：拥有一个 Leader 所有的权限，并且可以维护整个产品。



图 2-11　用户信息

详细情况请参见表 2-1 和图 2-12。

表 2-1　用户角色及相应职能

角　色	权 限 列 表	能　力
Guest	查看测试规范（组件、分类和测试用例的数据），查看关键字，查看度量	只能浏览数据
Test Executor Tester	执行测试用例，查看度量	只能执行测试
Test Analyst Senior tester	执行测试用例，查看度量，创建构建，查看、修改测试规范（组件、分类和测试用例的数据），查看关键字，创建、编辑、结合和删除测试需求，查看测试需求	编辑测试规范和执行需求
Test Designer	查看度量，查看、修改测试规范（组件，分类和测试用例的数据），查看关键字，创建、编辑、结合和删除测试需求，查看测试需求	编辑测试规范和测试需求
Test Leader	执行测试用例、创建构建、查看度量，创建、编辑、删除测试计划，设置风险/所有权，编辑/删除里程碑，编辑用例集，设置查看项目的权限，查看/修改关键字，查看/修改测试需求	拥有所有测试计划权限，编辑测试规范和执行测试
Administator	执行测试用例、创建构建、查看度量，创建、编辑、删除测试计划，设置风险/所有权，编辑/删除里程碑，编辑用例集，设置查看项目的权限，查看/修改测试规范（组件、分类和测试用例的数据），查看/修改关键字，查看/修改测试需求，创建、编辑和删除产品，创建、删除和维护用户	进行一切可行性操作，只有此用户可以维护产品和用户

同时，TestLink 初始配置支持不同地域用户对不同语言的需求，为用户提供不同的语言支持。

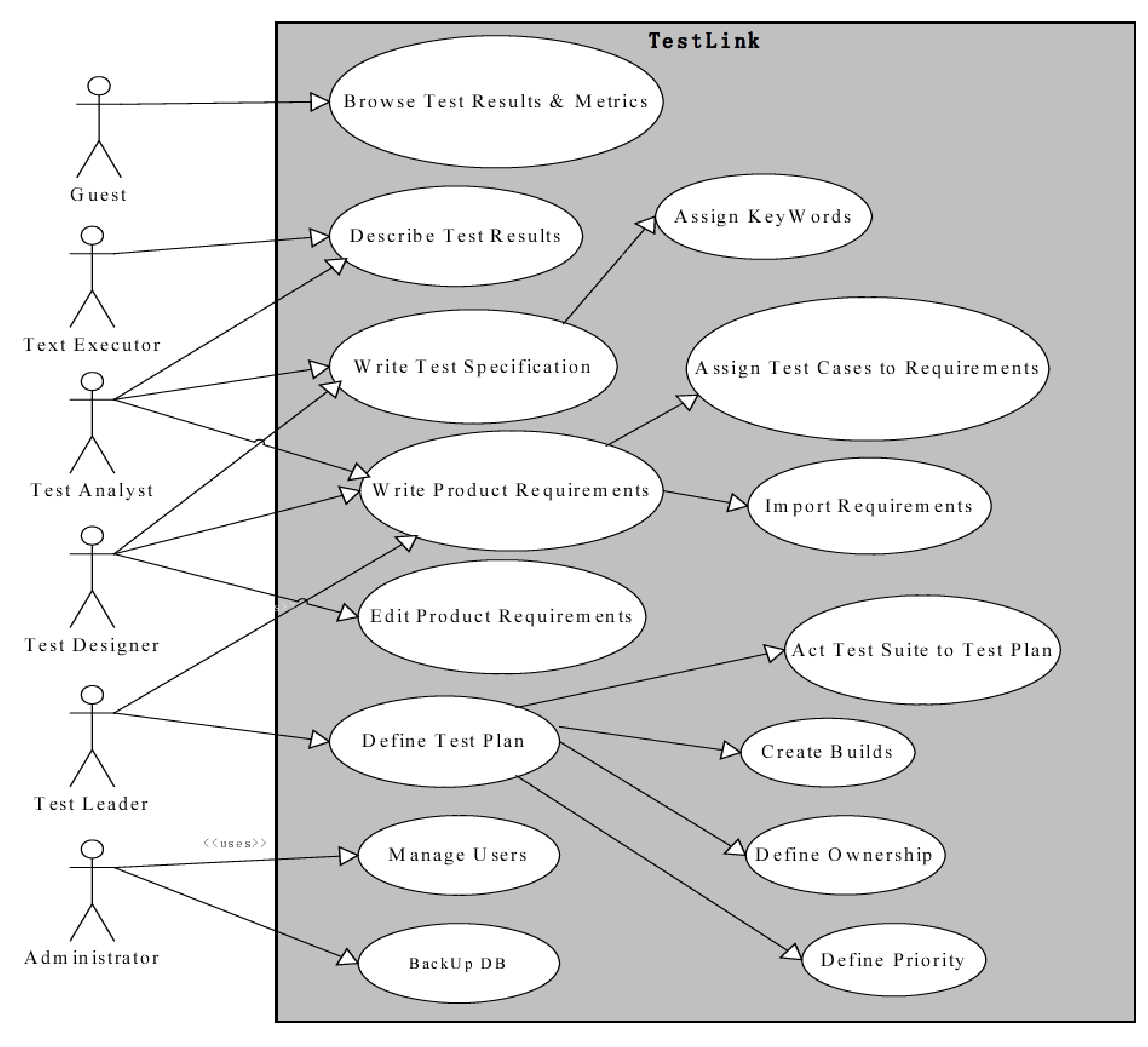


图 2-12　用户角色分类

3) 产品设置

创建一个新产品需要有管理员的权限，而且不能创建相同名称的产品，但为了使其显示得更清晰，可以为产品分配一个背景颜色，如图 2-13 所示。

创建一个新产品时要注意以下几点。

(1) 从系统中删除产品本身是不被认可的，因为产品的删除会使很多测试用例处于孤立的状态或者导致这些测试用例也被删除。

(2) 测试计划在特定时间里描绘产品的测试。这句话的意思就是说所有的测试计划需要根据产品测试用例来创建。

(3) TestLink 可以把数据输入到一个产品中，数据以 CSV 形式读入并在输入环节被进一步说明。

(4) TestLink 可以对多个产品进行管理，Admin 对产品进行设置后，测试人员就可以进行测试需求、测试用例、测试计划等相关管理工作了。TestLink 支持对每个产品设置不同的背景颜色，以便于产品的管理。

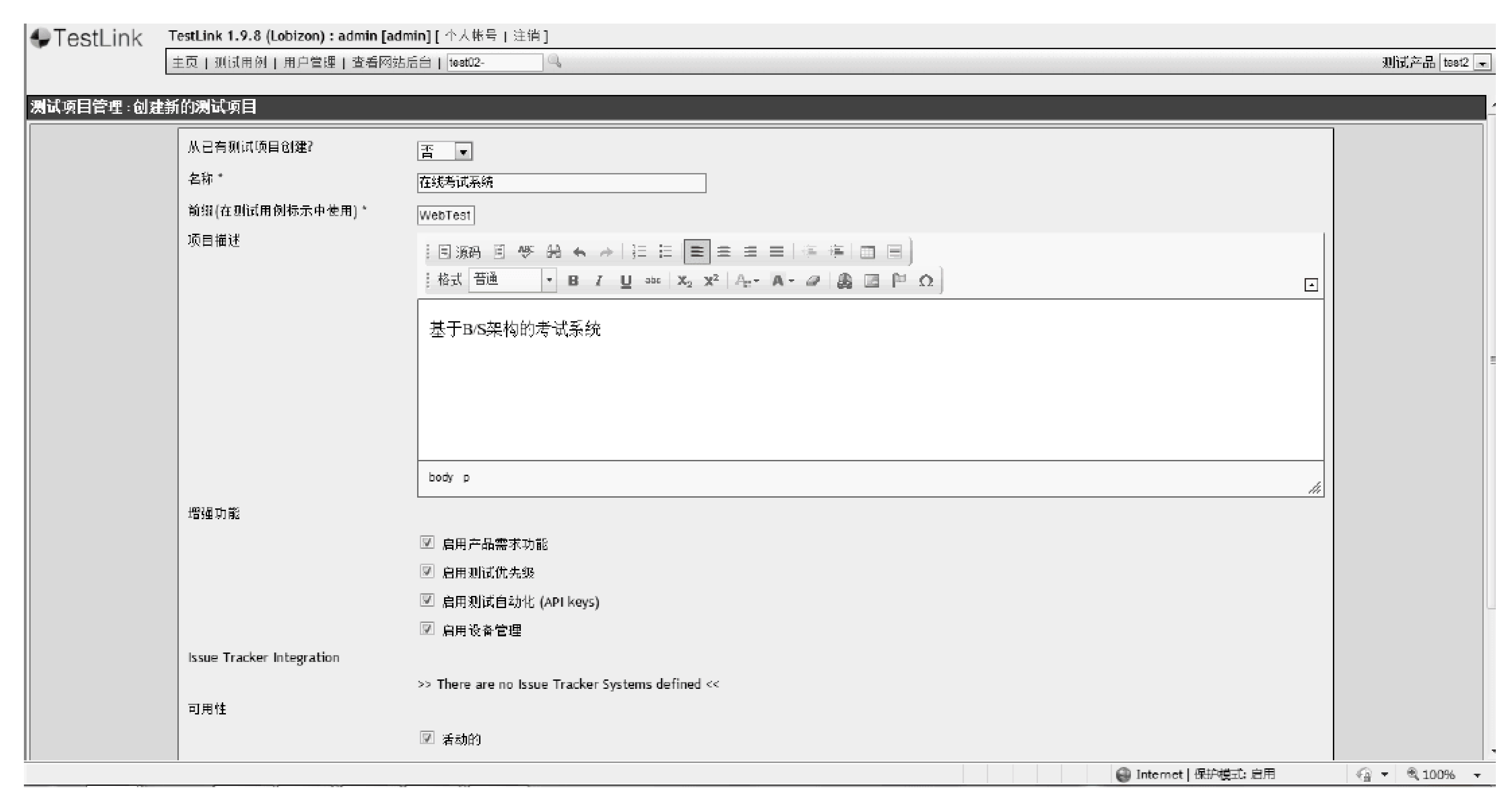


图 2-13　产品管理

2. 确定测试需求

为了验证一个系统是按照指定要求建立的，测试人员需要将测试与需求对应起来。对于每一个需求，可以设计一个或更多个测试用例。在测试执行的最后阶段，测试经理汇报测试的执行情况以及需求的覆盖率。基于这个汇报信息，客户和投资人决定系统是继续测试还是投入使用。为了保证系统是按照指定要求建立的，测试经理必须将风险和测试需求结合起来统筹考虑，以保证系统是按照客户和投资人的指定要求建立的。这样做有下列好处。

(1) 风险和需求相结合可以揭露潜在的或遗漏的需求，这对高风险系统来说尤为有意义。

(2) 测试可以首先集中在一个信息系统中的最重要的部分，首先涵盖最高优先权的风险。

(3) 促使测试人员按照客户和投资人的眼光来看待问题。这使得报告测试项目的状况更加容易。除此之外，还将有充足的依据决定是否进行更多测试还是冒点儿风险。

(4) 风险和它们的优先权使测试项目在面对压力时的谈判更加容易。例如，在这个测试项目之内，什么风险必须覆盖，哪些可以被延期。将风险与需求相结合进行测试能更加有效地控制测试项目，还能改善客户与股东之间的沟通。测试经理首先测试最高优先权的风险，测试过程将更有效，结果也更可靠。

1) 创建需求规格

单击主页，在主页上找到产品需求，新建一个需求规格。对于需求规格的描述比较简单，内容包含文档 ID、标题、范围，如图 2-14 所示。

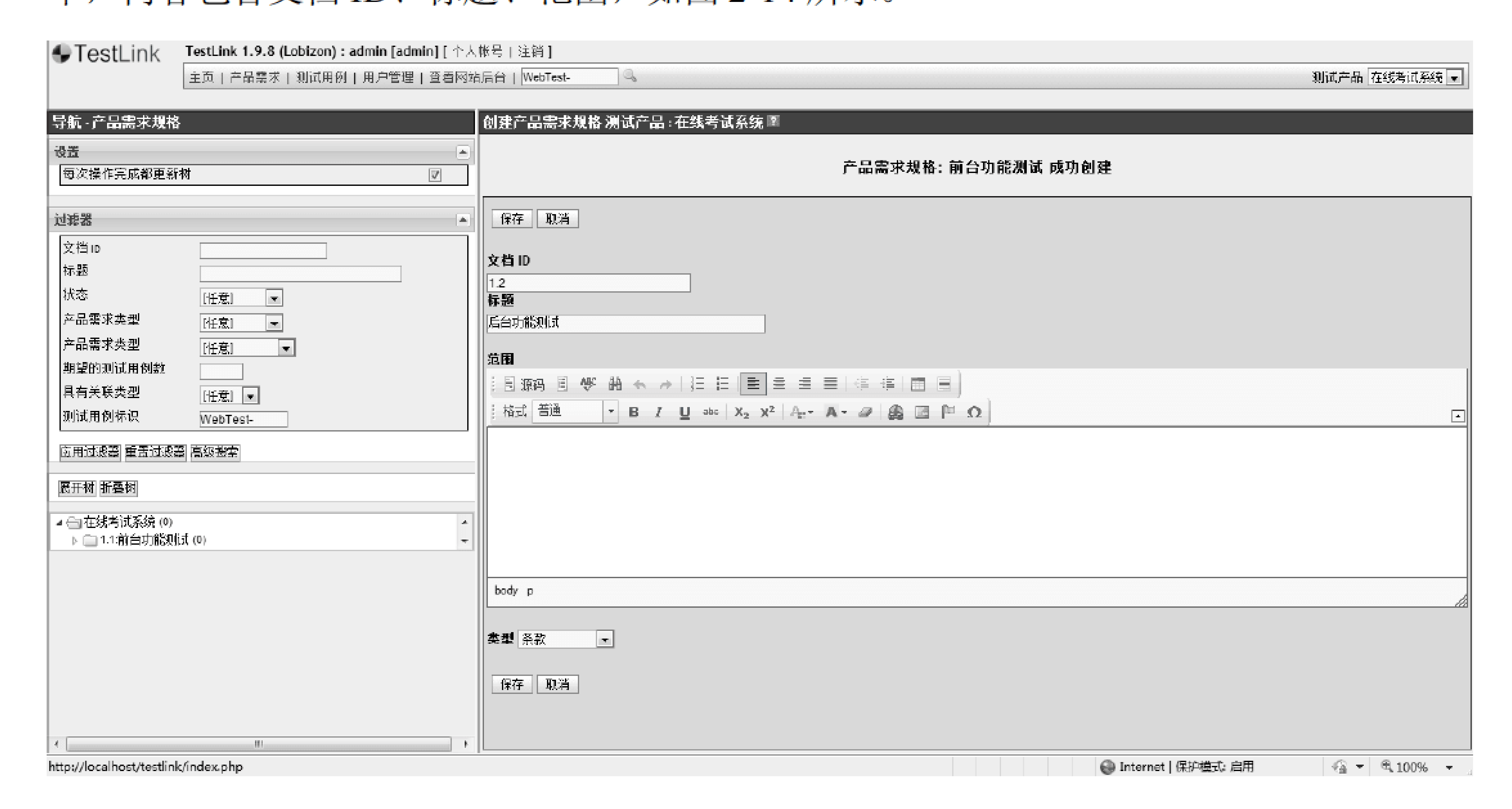


图 2-14　创建产品需求规格

单击左侧的“在线考试系统”节点。在右侧的界面中单击“新建产品需求规格”按钮。输入文档 ID“1”，标题“前台功能测试”。然后创建另一个规格文档 ID“2”，标题“后台功能测试”。完成以后如图 2-15 所示。



图 2-15　产品需求规格

2) 创建需求

选择要编辑的需求规格，单击该页面上的“创建新产品需求”按钮，开始新建测试需求。单击需求规格“1：前台功能测试”，添加它的需求：1.1 登录验证，需要的测试用例数一个，如图 2-16 所示。

文档标识
1.1
标题
登录验证
范围
源码
格式　普通
登录验证
body p
状态　草案
类型　用例
需要的测试用例数　1
保存　取消

图 2-16　创建需求

添加其他需求，如表 2-2 所示。

表 2-2　所有需求

需　　求	测试用例数
前台功能测试——登录验证	1
前台功能测试——学生注册	1
前台功能测试——成绩查询	1
前台功能测试——在线考试	3
前台功能测试——答题	1
后台功能测试——查询考生成绩	2

完成后的需求如图 2-17 所示。

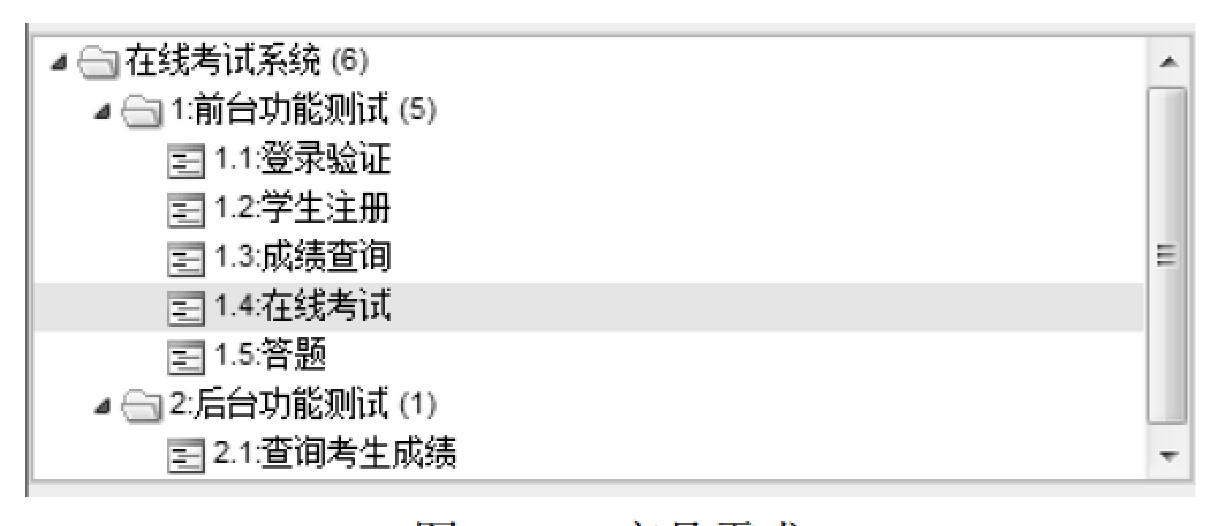


图 2-17　产品需求

测试需求内容包含：文档标识、名称、范围、需求的状态，以及覆盖需求的案例。TestLink 提供了两种状态来管理需求：合法的(valid)、不可测试的(not testable)。

TestLink 提供了从文件导入测试需求的功能，支持的文件类型有 CSV、CSV(DOOR)和 XML 三种。同时 TestLink 也提供了将需求导出的功能，支持的文件类型有 CSV 和 XML 两种。

TestLink 还提供了上传文件的功能，可以在创建测试需求的时候，为该需求附上相关

的文档。

3. 测试用例管理

TestLink 支持的测试用例管理包含两层：测试用例集或测试套件(Test Suites)、测试用例(Test Cases)。可以把测试用例集对应到项目的功能模块。测试用例与每个模块的功能相对应。

可以使用测试用例搜索功能从不同的项目以及成百上千的测试用例中查到需要的测试用例，甚至可以直接将别的项目里的测试用例复制过来，这样就解决了测试用例的管理和重用问题。但是，还有一个问题没有解决，那就是与测试需求的对应问题。在测试管理中，测试用例对测试需求的覆盖率是人们非常关心的，从需求规格说明书中提取出测试需求之后，TestLink 会提供管理测试需求与测试用例间对应关系的功能。

1) 创建用例集

单击主页上的“测试用例”菜单下的“编辑测试用例”，出现如图 2-18 所示界面。

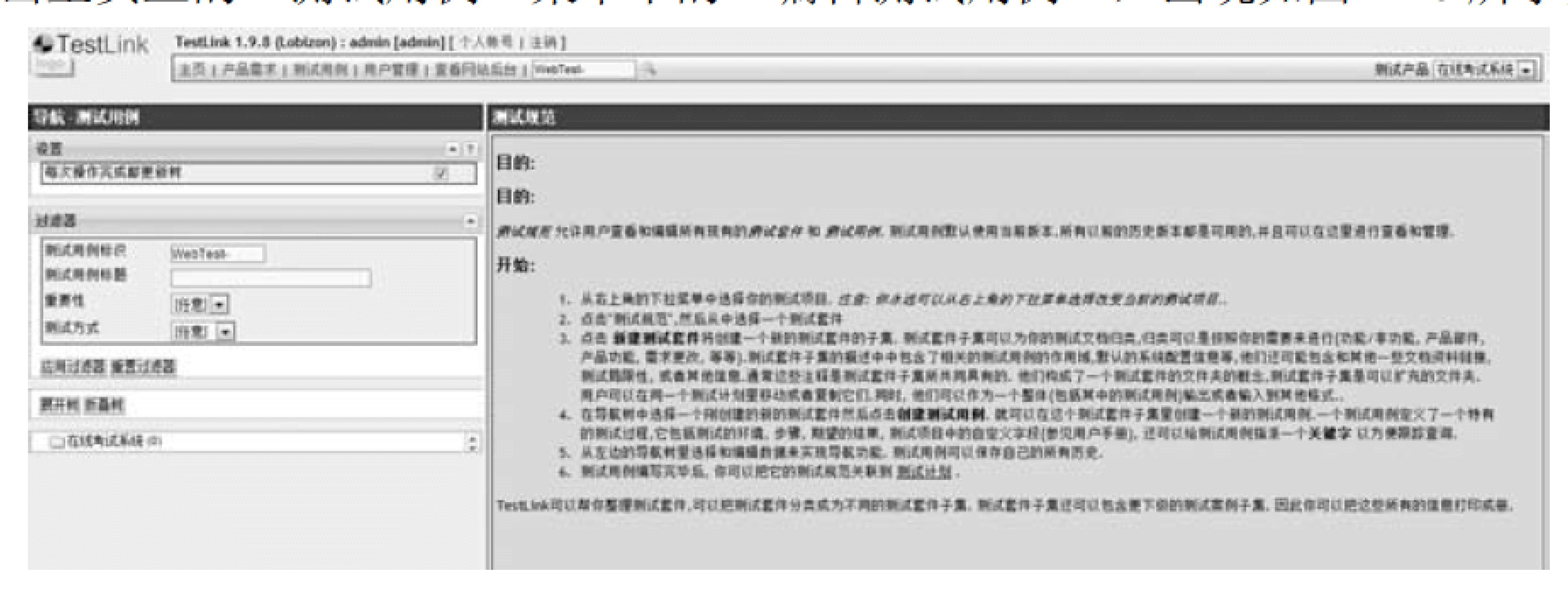


图 2-18　创建用例集主页面

单击左侧的“在线考试系统”，在右侧窗口单击“新建测试用例集”按钮，添加两个测试集合“前台功能测试（学生端）”和“后台功能测试（教师端）”，如图 2-19 所示。



图 2-19　创建用例集

填写好相关的内容后，可以单击“创建测试用例集”按钮，创建该用例集。完成以后如图 2-20 所示。



图 2-20　测试用例集创建完成

2) 添加测试用例

选择创建好的测试用例集“前台功能测试（学生端）”，单击该页面右侧的“创建测试用例”按钮，新建一个名为“登录验证”的测试用例，如图 2-21 所示。

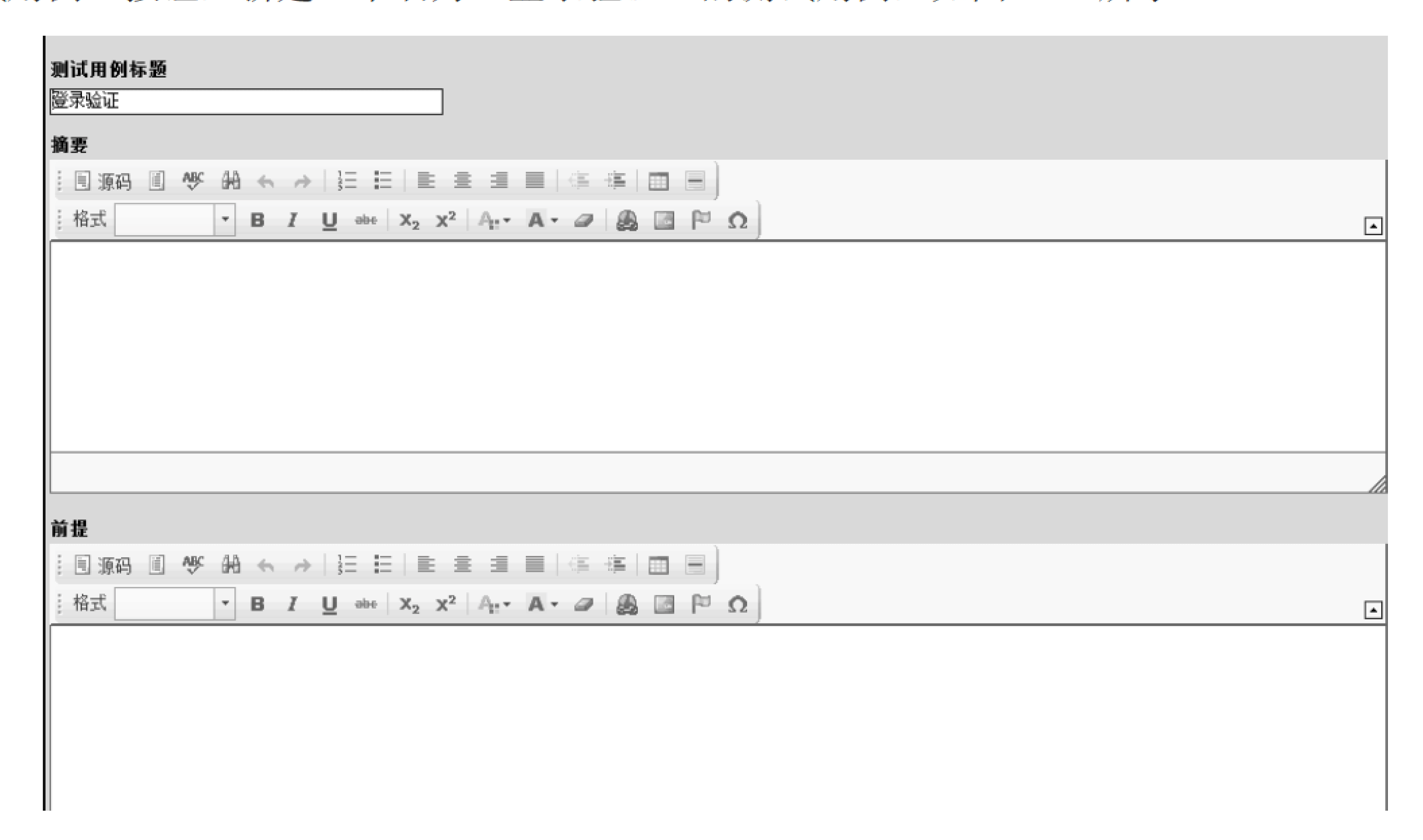


图 2-21　创建测试用例

测试用例创建成功后，单击“创建步骤”按钮，输入数据，单击“保存”按钮，如图 2-22 所示。



图 2-22　创建测试步骤

按照此方法添加如表 2-3 所示的数据。

表 2-3　所有测试数据

测试用例	步　骤　动　作	期望的结果
登录验证	1. 登录用户名为空，密码不为空 2. 登录用户名不为空，密码为空 3. 输入错误的用户名或密码 4. 输入正确的用户名和密码	1. 提示用户名或密码不能为空 2. 提示用户名或密码不能为空 3. 提示用户名或密码错误 4. 显示登录成功的界面
学生注册	1. 输入的任何数据项为空 2. 输入的密码位数少于 6 位	1. 提示该项不能为空 2. 提示密码至少 6 位
成绩查询	1. 单击“查询成绩”按钮	1. 在页面中显示考生的成绩
答题	1. 选中正确的选项 2. 选中错误的选项	1. 在成绩中加上该题目的分数 2. 成绩无变化
在线考试——上一题	1. 单击“上一题”按钮	1. 显示上一题的内容
在线考试——下一题	1. 单击“下一题”按钮	1. 显示下一题的内容
在线考试——交卷	1. 单击“交卷”按钮	1. 计算考生的得分
后台功能测试——查询考生成绩	1. 输入考生姓名/学号，单击“查询”按钮 2. 不输入任何查询关键词，单击“查询”按钮	1. 显示该考生的考试成绩信息 2. 显示所有考生的考试成绩信息

测试用例包含如下部分：标题，要求是简短的描述语或缩写词(如 TL-USER-LOGIN)；摘要，要求短小、概括全面；步骤，描述测试说明(输入行为)以及可以包括的前提条件及清除信息；期望的结果，描述检验点和一个测试产品或系统的预期行为。

建议：在编写测试用例时，要细分每一个数据类型。有些测试用例的编写步骤是相同的，变化的可能只是数据类型，可以采用复制的方法来实现。如果有多个分类下面的测试用例操作相同，而只是部分数据类型或字段名称不同，则可以通过移动测试用例的方法来减少测试用例工作量。同时，也可以在创建测试用例的摘要中，将不同的测试数据罗列，然后在测试步骤中，根据不同的测试数据，执行相同的操作。

完成上述操作后，创建好的测试用例树如图 2-23 所示。



图 2-23　测试用例树

3) 删除测试用例

经过主管许可后，通过“删除”按钮可以将测试用例集和测试用例从一个测试计划中删除。当第一次创建一个测试计划时，由于其中还没有包含任何结论，删除数据也许是有益的。但是在大部分情况下，删除测试用例会导致与它们相关联的所有结果都丢失。因此，使用这项功能时一定要十分谨慎。

4) 需求关联

测试用例和软件/系统需求之间是 n 对 n 的关系。对于一个产品来说，这个功能是一定要用到的。用户可以通过主页面中的指派产品需求，把需求指派给测试用例。

单击主页“产品需求”模块下的“指派产品需求”菜单，进入指派需求页面，选中左侧用例树中的测试用例，再选择右侧对应的测试需求，进行指派即可。本特性允许在需求和测试用例之间建立关系，设计人员可以定义“0..n”到“0..n”的关系。例如，一个需求可以被指派给零个、一个或多个测试用例上，反之亦然。

在线考试系统中测试用例与需求的关联如表 2-4 所示。

表 2-4　测试用例与需求的关系

测 试 用 例	需　　求
登录验证	登录验证
学生注册	学生注册
成绩查询	成绩查询
答题	答题
在线考试——前一题	在线考试
在线考试——下一题	
在线考试——交卷	
后台功能测试——查询考生成绩	查询考生成绩

完成上述的操作，可以查看已经指派的测试用例。单击顶部的“产品需求”栏，在所示页面中单击左下的需求“1.4：在线考试”，可以看到需求的覆盖率，如图 2-24 所示。

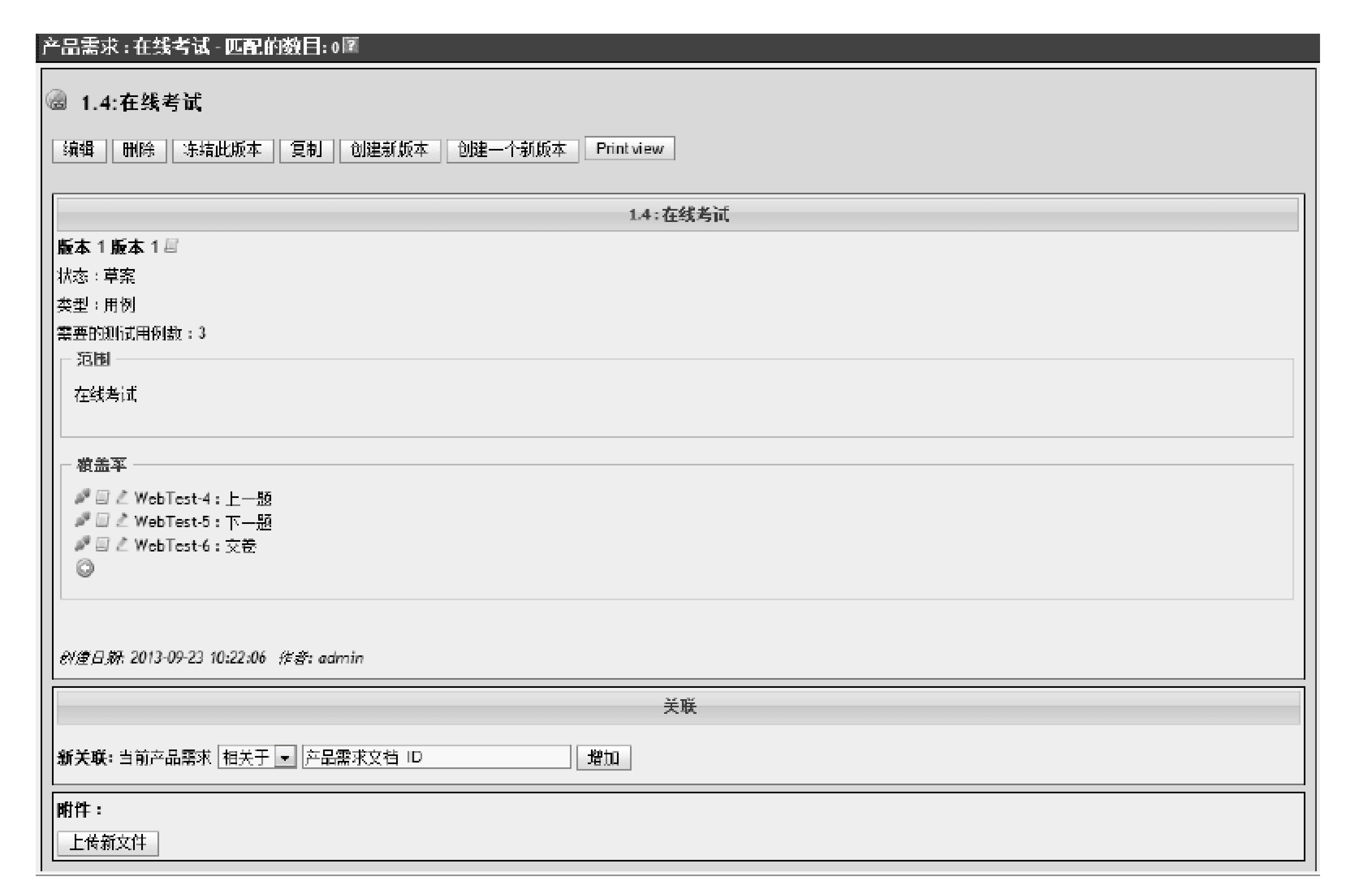


图 2-24　需求覆盖率

4. 制订测试计划

制订测试计划只能由 admin 用户进行。

1) 创建测试计划

测试计划是执行测试用例的基础，测试计划由测试用例组成，而这些测试用例是在特定的时间段里输入到产品中的。测试计划只能主管创建，但也可以从其他测试计划中产生，同时还允许用户在紧急情况下及时地从测试用例中创建测试计划。当为一个小模块创建一个测试计划时，上述都是可以使用的方法。为了使用户可以查看测试计划，此用户必须有足够的权限，而查看测试计划的权限要在定义用户/产品权限页面由主管分配，当用户被告知他们不能查看工作中的项目时，权限是需要记住的重要事情。

在 TestLink 系统中，一个完整的测试计划要包括各测试阶段的名称(如集成测试阶段、系统测试阶段)。

单击主页“测试计划管理”模块下的“测试计划管理”菜单。在出现的页面中单击“创建”按钮，进入测试计划创建页面，如图 2-25 所示。



图 2-25　创建测试计划

测试计划的内容包括：计划名称、计划描述，以及是否从已有的测试计划创建，如果选择从已有的测试计划中创建，则新创建的测试计划包含选择的已有测试计划的所有相关联的信息，比如已有测试计划分配的测试用例。

创建一个名为“在线考试系统——测试计划”的测试计划。

2) 创建测试里程碑(明确每个测试阶段的开始和截止时间，以及完成 A、B、C 三种优先级的比例)

单击主页面“测试计划管理”模块下的“编辑/删除里程碑”菜单，创建一个新的测试里程碑。测试里程碑的内容包括：名称、日期、优先级，如图 2-26 所示。

创建里程碑
名称　测试基本完成
日期 (YYYY-MM-DD)　2013-10-09　里程碑的日期必须大于等于今天
开始日期　2013-09-18　开始日期为选填
A优先级完成比例[0-100%]:　100
B优先级完成比例[0-100%]:　0
C优先级完成比例[0-100%]:　0
保存 取消
里程碑考虑一个特定时期的测试执行。
开始时间为开始日期 00:00:00如果没有指定开始日期，将覆盖所有执行 - 结束日期为目标日期 23:59:59.
所有目标日期之后的执行将被忽略。
里程碑将在所有不同优先级的"子里程碑"都完成时达成。里程碑的状态显示在一般测试计划进度中。

图 2-26　创建测试计划里程碑

3) 版本管理（Builds/Releases）

测试计划做好后，就应该制定版本，比如 ver1.0。如果测试过程中发现了 Bug，修改之后就产生了 ver2.0。这时应该追加版本，相应地接下来未完的测试以及降级测试都应该在新的版本上完成。所有测试完成后可以统计在各个版本上测试了哪些用例，每个版本上是否都进行了降级测试等。

单击主页“测试计划管理”模块下的“版本管理”菜单，创建一个新的测试版本，如图 2-27 所示。



图 2-27　定义测试的版本

测试版本的内容包括版本的标识、版本的说明以及选择该版本是否为活动和打开的版本。如果是活动的版本，说明该版本可以用，否则该版本不可用。如果是打开的版本，则该版本的测试结果可修改，否则不可修改。在 TestLink 中，“执行”由版本和测试用例组成。如果在一个项目中没有创建版本，执行页面将不允许执行，度量页面则完全是空白的，版本通常不能被编辑或删除。

4) 安排测试人员(定义用户/测试计划角色权限)

单击主页面“测试计划管理”模块下的“指派用户角色”菜单，为测试计划指派用户。测试用户如表 2-5 所示。

表 2-5　测试用户

账　　号	用 户 类 型	备　　注
Tester1	Tester	测试工程师 1
Tester2	Tester	测试工程师 2
TD1	TdESIGNER	测试组长
ST1	Senior Tester	资深测试工程师
Test_manager1	Leader	测试经理

在为测试计划指派用户页面，可以选择测试计划，选择好需要指派权限的测试角色后，单击“更改”按钮，则可以更改测试计划。选择好测试计划后，可以将该测试计划以不同的角色分配给不同的用户，通过角色列表，可以选择用户对该测试计划的操作角色，选择结束后，单击“更新”按钮，可以保存结果，如图 2-28 所示。



图 2-28　指派测试计划用户角色

5) 添加测试用例到测试计划

在主页通过测试计划下拉列表，先选择一个测试计划，单击“测试用例集”下的“添加/删除测试用例到测试计划”按钮，进入测试计划中添加测试用例。

可以将已经创建好的测试用例指派给该测试计划。单击一个测试用例集，可以看到该测试用例集下所有的测试用例。可以选择该测试计划中要执行的测试用例，单击“增加选择的测试用例”来添加或删除测试用例，可以将选择好的测试用例分配给该测试计划，如图 2-29 所示。

6) 移除测试用例

在如图 2-29 所示的页面中，可以在复选框组中勾选不需要在该测试计划中执行的测试用例，然后单击“添加/删除选择的”按钮，将测试用例移除，如图 2-30 所示。

7) 设置测试用例的所用者（给测试人员分派测试任务）

单击主页面“测试用例集”模块下的“指派执行测试用例”菜单，进入指派测试用例页面，可以为当前测试计划中所包含的每个测试用例指定一个具体的执行人员。



图 2-29　添加测试用例到测试计划



图 2-30　移除测试用例

在指派测试用例页面，从左侧用例树中选择某个测试用例集或测试用例，右侧页面将会出现下拉列表以供选择用户。选择好合适的用户后，选中测试用例前面的复选框，单击“保存”按钮即可完成测试用例的指派工作，如图 2-31 所示。



图 2-31　设置测试用例的所用者

当然，这里也可以进行批量指定——右侧页面的最上方有一个下拉列表可以选择用户，在下方的测试用例列表中选择要指派给该用户的用例，然后单击后面的“执行”按钮即可完成将多个用例指派给一个人的操作。

8) 执行测试

在 TestLink 顶部的菜单栏中有“执行”选项，单击进入“测试用例执行”页面，如图 2-32 所示。

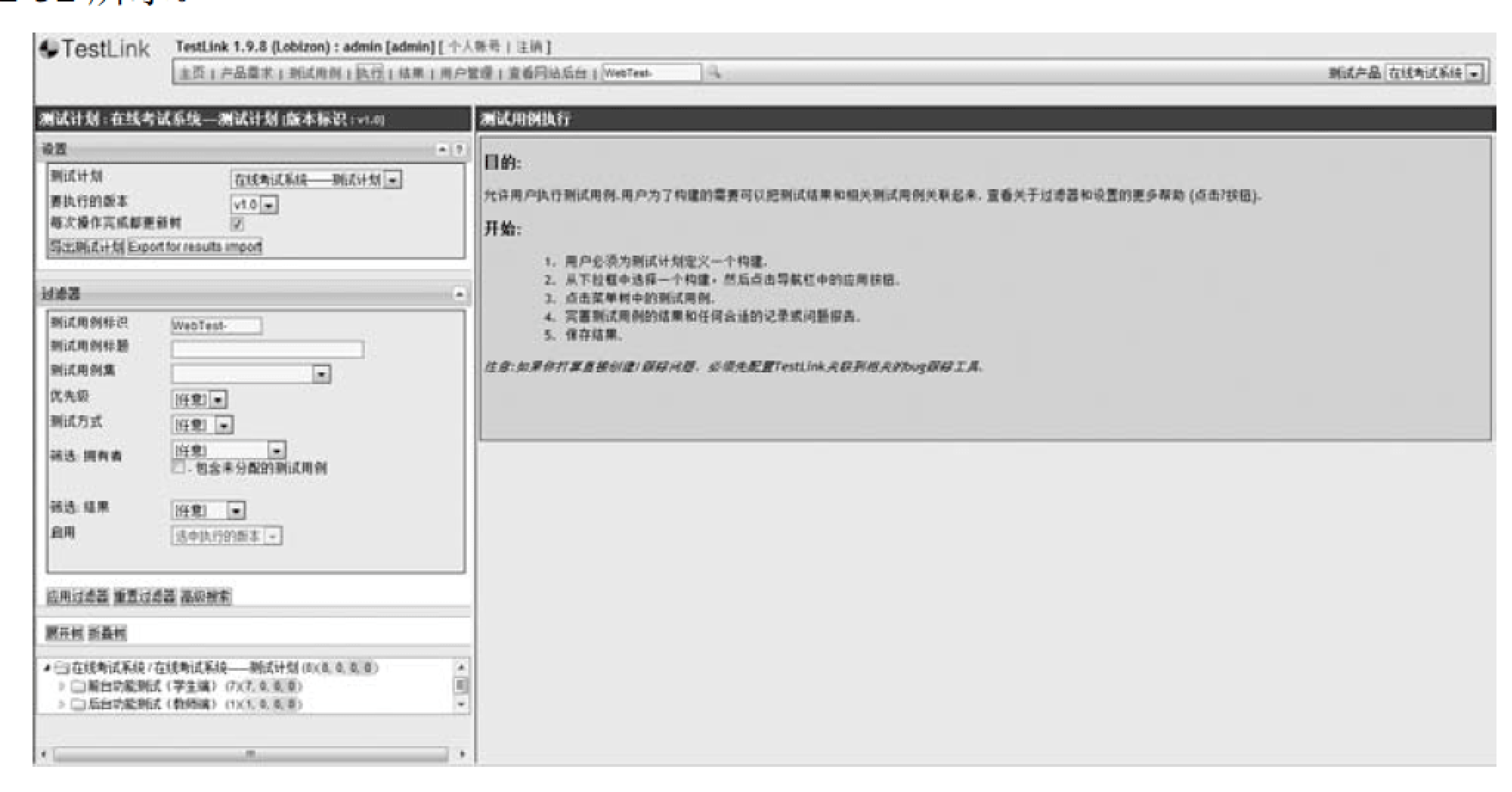


图 2-32　“测试用例执行”页面

这里需要说明一点，虽然“执行”表面上针对的是测试计划，但实际上对应的是测试计划中测试用例的执行情况。

在左侧用例树中，可以根据具体的条件来选择测试用例。选择某一个测试用例集后，页面右侧上方会出现测试计划、build 描述、测试集说明等信息，页面下方则是每个测试用例的详细情况，同时每一个测试用例的最后部分，有“说明/描述”对话框，可以在这里输入执行的一些说明性情况，还有“测试结果”，这两个对话框都是需要执行完测试用例后自己来填写的。

测试结果分为以下 4 种情况。

(1) 通过：该测试用例执行通过。

(2) 失败：该测试用例没有执行成功，这个时候可能就要向 Mantis 提交 Bug 了。

(3) 锁定：由于其他用例执行失败，导致此用例无法执行，被阻塞。

(4) 尚未执行：如果某个测试用例没有执行，则在最后的度量中将其标记为“尚未执行”。

进行测试的页面如图 2-33 所示。

5. 测试报告

执行用例的过程中一旦发现 Bug，需要立即把其报告到 Bug 管理系统 Mantis 中去。TestLink 提供了与多种 Bug 跟踪系统关联的接口配置，目前支持的 Bug 管理系统有 Jira、Bugzilla、Mantis。与 Mantis 集成的方法如下。

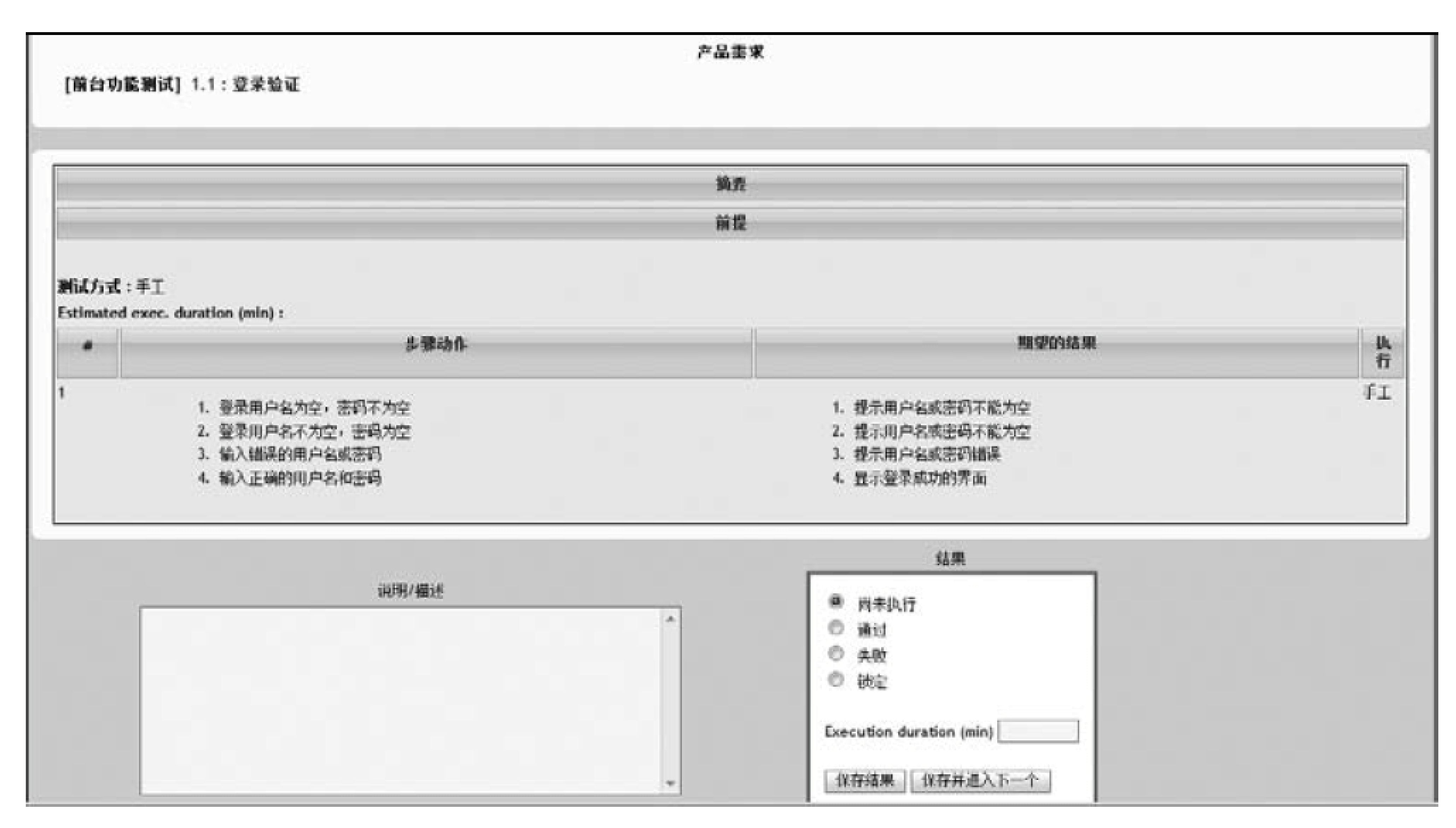


图 2-33　进行测试

(1) 主页→System→Issue Tracker Management，单击“创建”，添加一个 Issue Tracker，如图 2-34 所示。（单击 Show configuration example 可弹出示例，如果不能弹出，去 lib/issuetrackerintegration/目录中相关文档中找 public static function getCfgTemplate()函数。）

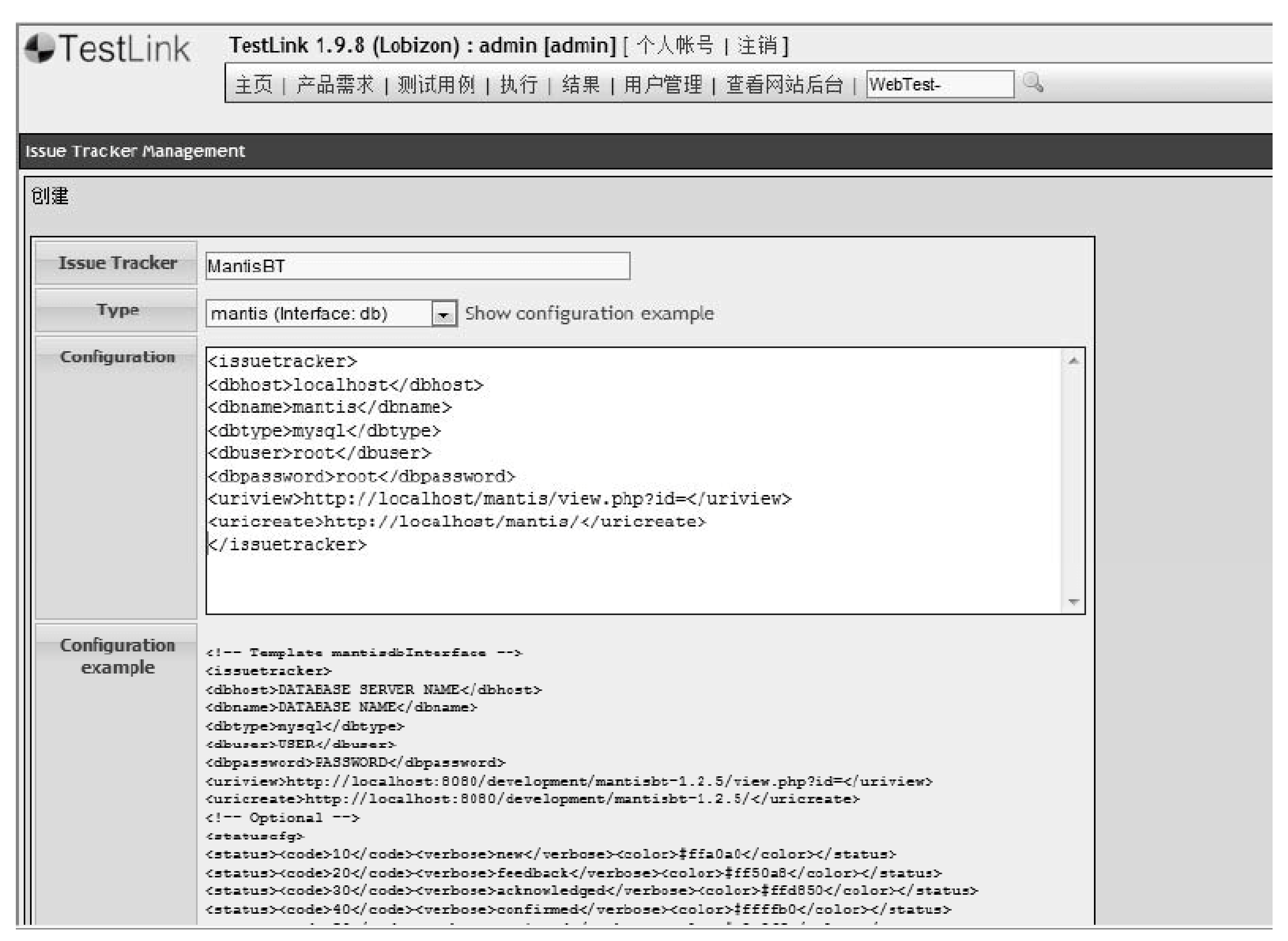


图 2-34　添加 Issue Tracker

(2) 主页→产品管理→测试项目管理，在项目编辑页面选择缺陷跟踪系统，如图 2-35 所示。

(3) TestLink 和 Mantis 集成后，执行完测试，测试结果中会多出一项 Bug 管理的项，在这个记录后面会有一个类似小虫子的标记，单击这个小虫子标记后，会出现一个记录 Bug 号的输入框，如图 2-36 所示。如果测试用例是失败的，则可以在这个地方输入该测试用例所发现 Bug 在 Mantis 中的 ID，然后该记录下面会出现一个 ID 链接，如图 2-37 所示，单击该 ID 链接后，可以直接链接到 Mantis 中该 Bug 的页面，如图 2-38 所示。

图 2-35　选择缺陷跟踪系统



图 2-36　添加 Bug



图 2-37　与 Manits 关联的链接

6. 测试结果分析

TestLink 根据测试过程中记录的数据，提供了较为丰富的度量统计功能，可以直观地得到测试管理过程中需要进行分析和总结的数据。单击首页导航菜单栏中的“结果”菜单，即可进入测试结果报告页面，主要包括的功能如图 2-39 所示。

1) 总体测试计划进度

查看总体的测试情况，可以根据测试组件、测试用例拥有者、关键字进行查看，如图 2-40 所示。



图 2-38　Mantis 中的 Bug 页面



图 2-39　测试结果报告主要功能



图 2-40　总体测试计划进度

2) 失败的测试用例

统计所有当前测试结果为失败的测试用例。

3) 锁定的测试用例

统计所有当前测试结果为锁定的测试用例。

4) 尚未执行的测试用例

统计所有当前尚未执行测试的测试用例，如图 2-41 所示。

没有运行测试用例

测试产品：在线考试系统
测试计划：在线考试系统——测试计划

打开/折叠组　显示全部列　复位到默认状态　刷新　重置过滤器　多列排序

版本标识：v1.0 (6 Items)

测试用例集	测试标题	版本	指派到	摘要
后台功能测试（教师端）	WebTest-7:查询考生成绩	1	Tester2	
前台功能测试（学生端）	WebTest-3:成绩查询	1	Tester1	
前台功能测试（学生端）	WebTest-8:答题	1	Tester1	
前台功能测试（学生端）/在线考试	WebTest-4:上一题	1	Tester1	
前台功能测试（学生端）/在线考试	WebTest-5:下一题	1	Tester1	
前台功能测试（学生端）/在线考试	WebTest-6:交卷	1	Tester1	

Attention: This report considers ONLY test cases with tester assignment
此报告显示测试计划中所有未被执行的测试用例

由TestLink 生成 2013-10-07 08:32:02

图 2-41　尚未执行的测试用例

5) 图表

单击图表，可以看到 TestLink 以图表的形式生成的报告，非常直观。这里主要是通过图表的形式来表示测试用例的执行情况，红色表示测试失败，蓝色表示锁定用例，绿色表示通过测试，黑色表示尚未执行，如图 2-42 和图 2-43 所示（参见彩图）。

注意： *TestLink 图表中文显示乱码的解决办法如下。*

(1) 从 Windows 字体库中找到 SIMYOU.TTF（幼圆），将其复制到 TestLink 中的 pchat 字体目录。路径为：E:\xampp\htdocs\testlink\third_party\pchart\Fonts（以本书安装路径为例）。

(2) 修改 config.inc.php 文件，将 $tlCfg->charts_font_path =TL_ABS_PATH."third_party/pchart/Fonts/tahoma.ttf";中的字体 tahoma.ttf 修改为 SIMYOU.TTF。

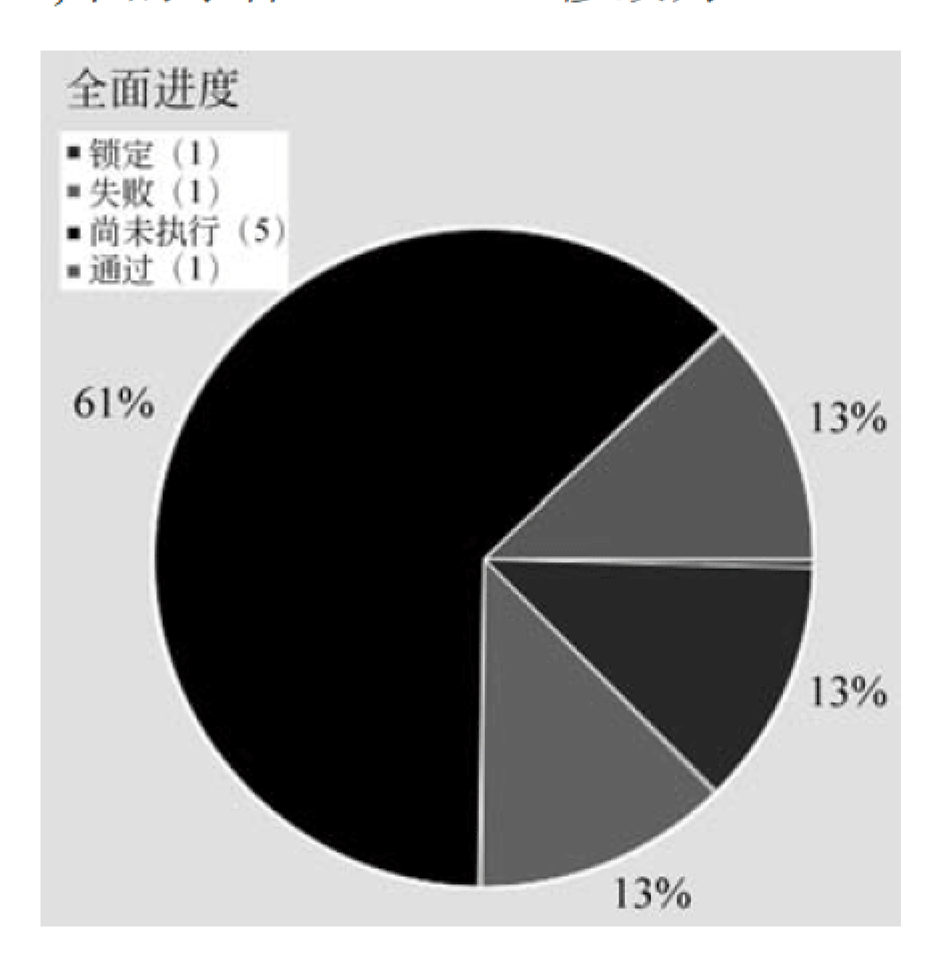


图 2-42　总体测试结果饼图

总地来说，TestLink 已经创造了很好的测试管理条件，其配置也相对简单，可以根据自己的需要进行相关的配置。另外，TestLink 是开源的测试工具，这也提高了灵活性。不过在 TestLink 的使用过程中，用户也感觉到了一些不便，比如很多情况下，需要回到主页面才能单击一些链接，并且 TestLink 和缺陷管理工具的整合需要手工来完成。另外，一些描述信息需要用文字描述，如果能提供填表的方式，效果会更好。

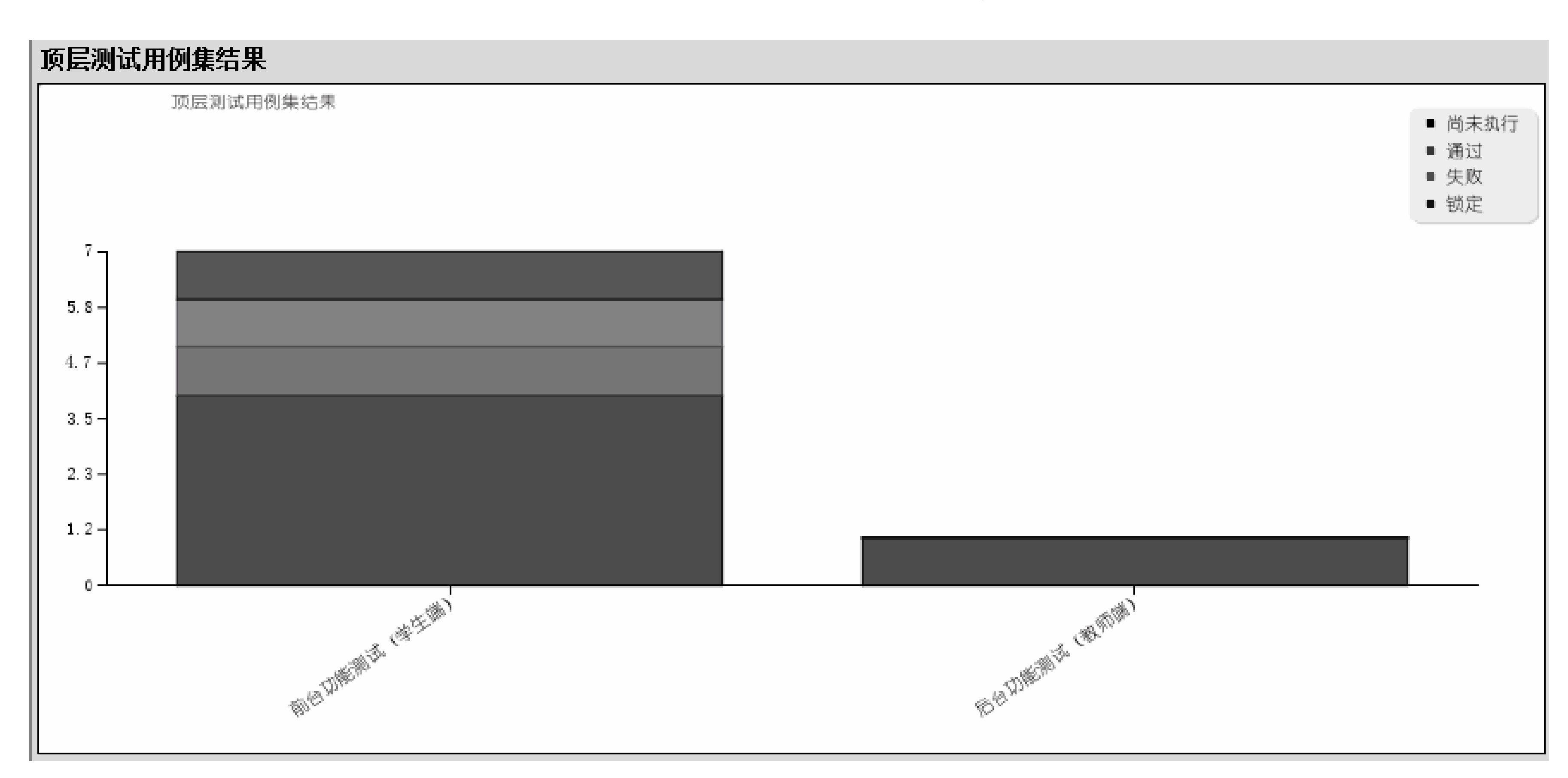


图 2-43　顶层测试用例集结果

实 验 习 题

1. 建立 TestRunner(Testopia)的测试管理环境，尝试与相关的缺陷管理工具集成，并与 TestLink 进行功能比较。

2. 建立一个测试管理与缺陷管理的环境，并通过一个具体实例，来完整地说明测试管理与缺陷管理的流程。

第Ⅱ部分　静态分析篇

程序静态分析是在不执行程序的情况下对其进行分析的技术，简称为静态分析。大多数情况下，静态分析的输入都是源程序代码，只有极少数情况会使用目标代码。

静态分析工具扫描所测试程序的正文，对程序的数据流和控制流进行分析，然后给出分析报告。通常采用以下方法进行源程序的静态分析。

1. 生成各种引用表

(1) 直接从表中查出说明/使用错误等，如循环层次表、变量交叉引用表、标号交叉引用表等。

(2) 为用户提供辅助信息，如子程序(宏、函数)引用表、等价(变量、标号)表、常数表等。

2. 编程规范检查

检查被分析程序违反编程标准的错误，如模块大小、模块结构、注释的约定、某些语句形式的使用，以及文档编制的约定等。

3. 静态错误分析

静态错误分析主要用于确定在源程序中是否存在某类错误或“危险”结构。

(1) 类型和单位分析：为了强化对源程序中数据类型的检查，发现数据类型上的错误和单位上的不一致性，在程序设计语言中扩充了一些结构。如单位分析要求使用一种预处理器，它能够通过使用一般的组合/消去规则，确定表达式的单位。

(2) 引用分析：使用最广泛的静态错误分析方法就是发现引用异常。如果沿着程序的控制路径，变量在赋值以前被引用或者变量在赋值以后未被引用，那么就会发生引用异常。为了检测引用异常，需要检查通过程序的每一条路径，也可以建立引用异常的探测工具。

(3) 表达式分析：对表达式进行分析，以发现并纠正表达式中出现的错误。包括：在表达式中因不正确地使用了括号而造成的错误、数组下标越界造成的错误；除式为零造成的错误；对负数开平方或者对 π 求正切值造成的错误。表达式分析还包括对浮点数计算的误差进行检查。

(4) 接口分析：接口的静态错误分析主要是检查过程及函数过程之间接口的一致性。因此要检查形参与实参在类型、数量、维数、顺序、使用上的一致性，以及全局变量和公共数据区在使用上的一致性。

(5) 逻辑分析：检查逻辑上可能存在错误的结构以及多余的不可达的程序段，如交叉转入转出的循环语句，为循环控制变量赋值，存取其他模块的局部数据等。

4. 静态特性的统计功能

(1) 程序编写信息统计，包括各种类型源语句的出现次数，标识符使用的交叉索引，标识符在每个语句中使用的情况，函数与过程的引用情况，任何输入数据都执行不到的孤立代码段，未经定义的或未曾使用过的变量，违背编码标准之处，以及公共变量与局部变量的各种统计。

(2) 用来做错误预测和程序复杂度计算，如注释率、McCabe 圈复杂度，以及基于操作符和操作数统计的 Halstead 科学度量法等。

静态分析工具能够保证代码的质量，发现并警告潜在的 Bug。静态分析的结果可作为静态测试的基础，用于进一步查错，并为测试用例的设计及选取提供指导依据。

目前，流行的独立分析工具有 C 的 lint 和 Smalltalk 的 lint 等，许多现代的 IDE 也同样能够对代码进行静态分析，还能够随代码的编辑进行增量的检查。

另外，支持程序走查或审查的程序理解工具也是静态分析工具中非常重要的一类工具，本篇还将对这些工具进行详细介绍。

第3章　程序理解工具

程序理解是人们将程序及其环境对应到面向人的概念知识的过程，它是软件开发过程中的一项重要活动，无论是软件的维护还是测试，抑或是逆向工程，都离不开对源代码的理解。

尽管程序理解可以手工进行，但要达到好的效果、高的效率，就必须要运用程序理解技术并在工具的支持下进行。随着软件规模及复杂度的不断增大，程序理解也变得越来越困难，需要耗费理解人员大量的时间和精力，却往往还不能得到理想的效果。因此，对通过计算机来完成软件系统分析和理解的程序理解辅助工具的需求变得越来越迫切。

3.1　程序理解概述

对于软件系统特别是大型复杂软件系统来说，由于分析和理解的困难性，导致其系统维护或系统演化任务异常艰巨，且成本开销巨大。根据 Boehm 等人的研究统计，软件维护开销占系统总成本的 50%～80%，而维护开销的 47%～62%则用于对软件系统的分析与理解。

3.1.1　程序理解的概念

程序理解是从计算机程序中获取知识信息的过程。这些知识信息可用于程序排错、程序增强、程序重用以及文档整理等方面的工作。程序理解的目标是从不同抽象层次，多视角、多方面地综合表达并展示程序理解结果，以加速对软件系统的理解。

程序理解是软件工程领域的一个重要部分。软件工程最为关心的是如何提高软件开发效率和软件产品质量，但是，就目前实际情况来看不容乐观。软件开发精力大多都花费在维护老系统上，而不是开发新系统上，用于维护、逆向工程或再工程方面的资源和时间占了 50%~70%。而软件维护过程的绝大多数时间被用于理解目标系统，国内外最新研究结果表明，维护和逆向工程工作的 70%的时间花在对系统的理解上。因此，程序理解不但是软件维护中的一个重要部分，而且还在整个软件维护、逆向工程或再工程过程中起到了举足轻重的作用。另外，程序理解对静态测试中的代码检查(如代码走查、代码评审等)能够起到有效的支撑作用。

程序理解旨在理解一段现存的程序，通过不断从源程序中抽取所需信息，检查部分代码，来逐步构建所需的理解。理解过程中，需要不断地从中获取知识并提炼。程序理解是一个复杂的过程，它需要分析目标系统，标识目标系统组件及其相互关系，创建不同形式或更高抽象层次的系统表示。程序理解的目标是理解软件系统以促使性能提高、纠错、建档、再设计或使用另外一种语言重新编程。不论是对目标系统进行概念建模、数据抽取还是系统抽象，都主要依赖于通过分析程序源代码来抽取程序结构和控制流信息。因此，程序理解是软件逆向工程主要的实现手段和行为活动，它贯穿于整个软件逆向工程，并且是决定软件逆向工程成败的关键。

3.1.2　程序理解的任务与内容

程序理解的任务是在不同的抽象级别上建立基本程序的思维模型，实际上就是建立从问题(应用)领域到程序设计(实现)领域的映射集。其范围包括从代码本身的模型到基本应用领域的模型，其目的是为了便于软件的维护、进展和再工程。从概念上讲，可以从抽象程度的几个不同层次来理解一个程序。归纳起来，有 4 个抽象程度不同的层次：实现层、结构层、功能层和领域层。

实现层是从程序设计语言的角度去对程序进行理解。检查单个的程序设计结构，程序按某种方式表示成抽象语法树、符号表或普通源文本。实现层包括程序扫描、语法提取、语义检查、静态分析、动态模拟运行等几个过程。

结构层是在程序语言的基础上，检查程序构造过程中的结构，对程序语言中所出现的各种实体(如全局变量、数据结构、过程等)以及它们之间的关系进行分析，如数据调用和控制流程图、程序调用图等，明确表示程序各组成部分之间的依赖关系。结构层包括逆向工程、信息提取、信息抽象、结构模型匹配、识别构造规范和结构、聚集分级结构等过程。

功能层是从程序中不同模块的功能来推断它们之间的逻辑联系。检查两个程序结构和行为(功能)之间的关系，同时研究程序构造的合理性。功能层包括设计恢复、语义和行为模式匹配等过程。

领域层是检查特定于应用领域的概念，进一步从功能上来推断此软件在其领域中的作用。领域层包括智能软件理解、格局识别、概念赋值和推理等过程。

以上 4 个层次是从一个抽象程度较低的层次(一般源代码)到一个抽象程度较高的层次的转换。这里主要介绍结构层的理解技术。

经过分析，全面、准确、迅速地理解程序，对于软件的开发、维护、质量保证及逆向工程和再工程有着重要意义。不过，程序理解的内容比较多，要讲述程序理解的内容，就必须结合具体的程序设计语言。尽管如此，程序理解一般都包含如下内容。

(1) 程序理解的功能和目标。

(2) 掌握程序的结构信息，即从程序中细分出若干结构成分，如程序系统结构、控制结构、数据结构、输入输出结构等。

(3) 了解数据流信息，即涉及的数据源于何处，在哪里被使用。

(4) 了解控制流信息，即了解每条路径的结果。

(5) 程序理解的操作(使用)要求。

(6) 面向对象的理解(对象、类、继承、通信等)。

3.1.3　程序理解的相关技术

结构层的理解技术有语句分析、表示程序单元之间关系的调用和被调用图、与程序内部结构相关的程序控制流图和数据流图等。

1. 语句分析

与自然语言类似，程序文件是由语句构成的，包括标识符、操作符、关键字、字符串、数字、标点符号等词法单元。语句分析分别构造程序的词法模型与语法模型，它们对应于词法分析与语法分析。由于这两种分析不生成任何语义信息，所以它们生成的模型仅适用于源代码的模式匹配。

2. 程序流分析

程序流分析技术，即在程序运行之前，通过静态分析去发现程序在运行行为方面的某些特性。程序流分析包括控制流分析和数据流分析两种，其中控制流分析侧重于对程序结构的分析，而数据流分析则侧重于对变量控制结构中数据的赋值、使用及传递情况的分析。

程序理解的首要任务是发现它的控制结构，即语句的可能执行路径，通过控制流分析建立过程内部的控制层次。控制流分析可以分为两大类：必经点分析和区间分析，这两种分析方法都是先对程序文本进行分析，将其转化成某种中间表示，然后在中间表示的基础上进行分析，具体选用哪种方法需根据程序理解的任务而定。

数据流分析是为了计算被分析程序在生成数据方面的行为，通常用于程序优化，即为程序优化建立环境。目前，使用最为广泛的方法是对控制流图进行循环分析，称为迭代数据流分析。

由于非结构化程序会给测试、排错和程序维护带来较大的困难，因此按照结构化程序设计的要求，理想的程序设计是尽量避免使用 goto 语句，程序的控制结构尽量做到单入口、单出口。基于这些原因，在对被测软件进行分析时，系统地检查程序的控制结构成了十分有意义的工作。

3. 软件结构图

软件结构图分为程序调用关系图(或函数调用关系图)和系统结构图。函数调用关系图或者说是程序调用关系图，都是对源程序中函数关系的一种静态描述，在函数调用关系图中，节点表示函数，边表示函数之间的调用关系。

系统结构图反映的是系统中模块的调用关系和层次关系，即谁调用谁，这里有一个时序关系。在系统结构图中，有向线段表示调用时程序的控制权将从调用模块转移到被调用

模块，并隐含了当调用结束时控制权将从被调用模块交回给调用模块。若一个模块有多个下属模块，那么这些下属模块的左右位置可能与它们的调用次序有关。

3.1.4　程序理解工具

阅读源代码是程序理解的一项重要活动，但是阅读别人的代码是枯燥乏味而且比较困难的工作，所以开发辅助工具成了程序理解的一项重要研究内容，并且在这一领域已经有了很多成果。这些工具能以更清晰、更可读、更可理解的方式组织和表示源代码，把人们从烦躁的代码阅读中解放出来。常见的辅助工具有以下几种：程序切分器、静态分析器、动态分析器等。程序切分器能够帮助程序员选择并只观察所提议更改影响的程序部件，不受无关部件的干扰，显示数据链和相关特征，使程序员能够跟踪更改影响。静态分析器能够帮助程序员快速提取模块、过程、变量、数据元素、对象与类、类层次结构等信息。在理解过程中，理解人员应该使用这些工具，以提高理解效率。

目前，除了针对 C++、Java 以及 Ada 等语言而专门开发的程序理解工具 Understand 外，专门用于程序理解的工具还不是很多，并且其中大多是作为辅助功能用于支持开发、测试或其他任务，如 Logiscope、Panorama++、McCabe IQ、Klocwork 以及有关的 IDE。国内北大青鸟在“九五”期间专门将 C++的程序理解工具作为科技攻关项目，并取得了较好的成绩，但遗憾的是并未看到他们广泛应用的商业化产品或开源产品。

令人感到欣慰的是，可以在网上找到几款不错的程序理解工具，尽管它们还存在不足或功能不全，但作为学习和实践之用还是足够的。

3.2　Oink 程序理解工具

Oink 是一个开源的、能够对 C 和 C++程序进行静态分析的工具，但它的基础或核心部分是程序的理解功能。

Oink 主要由 Scott McPeak、Karl Chen、Daniel S. Wilkerson 等人开发和维护。Oink 最基本的工具是 Cqual++，Cqual++的开发借鉴了开源工具 Cqual 的许多设计思想，并在其基础上进行了扩展和创新(基本上重写了代码)。Cqual 是由 Jeffrey S. Foster 等人开发的，可以对 C 程序进行基于类型(type-based)的数据流分析，而 Cqual++则可以对 C 和 C++程序进行基于类型的数据流分析。

Oink 的源码包(Oink-Stack)中有 smbase、ast、elkhound、elsa、libregion、libqual 和 platform-model 这几个部分。smbase、ast、elkhound 和 elsa 主要是由 Scott McPeak 开发的。

(1) smbase 包是由 Scott McPeak 开发的一个 C++的包和字符串库，用来代替 C++标准库的相关库，也可将此库用在别的项目开发中。

(2) ast 包是用于生成 AST(抽象语法树)的工具，它是 Oink 的可扩展的前端。

(3) elkhound 是用于管理 GLR 语法的分析器，它和 bison(一个用于自动生成语法分析器的程序)的功能相似，但 elkhound 还可以用来分析任何上下文无关的语法，而 bison 需要先生成 LALR(1)的上下文无关语法，然后再将其描述转换为可作语法分析的 C++程序(在新版的 bison 中已经加入了对 GLR 语法的分析)。

(4) elsa 包是由 Scott McPeak 开发的软件前端，它是在 elkhound 语法分析的基础上，将 C 和 C++程序生成为相应的 AST；elsa 也可以对程序进行部分类型检查，主要是与生成程序相关的部分，但不能期待它会做所有的类型检查工作。

(5) libregion 包是由 David Gay 开发的基于区域(region-based)的 C 语言内存管理库，这个库沿用了 Cqual 的部分。

(6) Libqual 包是由 Rob Johnson 开发的可序列化的、多态的类型修饰符推理接口，它在 Cqual 的基础上进行了部分的修改。

(7) platform-model 包是由 Karl Chen 开发的针对 C 库和 C++标准库程序的静态模型。

Oink 可以对 C 和 C++程序进行许多静态分析，主要是对数据流进行分析。在程序理解中，包括对源程序进行表达式级(expression-level)和类型级(type-level)的数据流以及语句级的(statement-level)控制流进行分析处理(这是通过 elsa 的相关功能实现的)，并可以实现对程序数据流图、控制流图、类继承关系图(C++)等的输出显示。

开发 Oink 主要是基于以下两个方面：一个是具有很快地查找程序错误和缺陷的能力，希望能和许多商业级的程序理解工具相媲美(在某些方面也的确做到了)；另一个是具有良好的可扩展性，可以比较方便地实现对工具前端和后端的裁剪和扩展。

Oink 在程序理解方面的应用主要是在结构层，即在程序语言的基础上，检查程序构造过程中的结构关系，对程序语言中出现的各种实体(包括全局变量、数据结构、过程等)以及它们之间的关系进行分析，如数据流图和控制流图等，给出程序各组成部分之间的依赖关系。

关于 Oink 的更多资料，可以去网站 https://github.com/dsw/oink-stack 查看。

3.2.1　Oink 环境建立

Oink 可以成功安装到一些版本的 Linux 系统、Apple OSX 系统以及 Linux 的虚拟机上。如果在 Windows 系统下安装了 cygwin，那么也可以在 cygwin 上安装并使用 Oink。

下面介绍 Linux 系统下 Oink 环境的建立(这里以在 VMWare 上安装 Fedora 8 系统为例来说明安装过程)。

(1) 下载 Oink 源代码。

免费下载网址：https://github.com/dsw/oink-stack。

(2) 安装 Oink。

在 Linux 系统终端上，进入解压缩后的安装文件目录，输入：

```
./configure && make clean all check
```

经过一段较长时间的等待后，可能会出现如图 3-1 所示的报错信息。

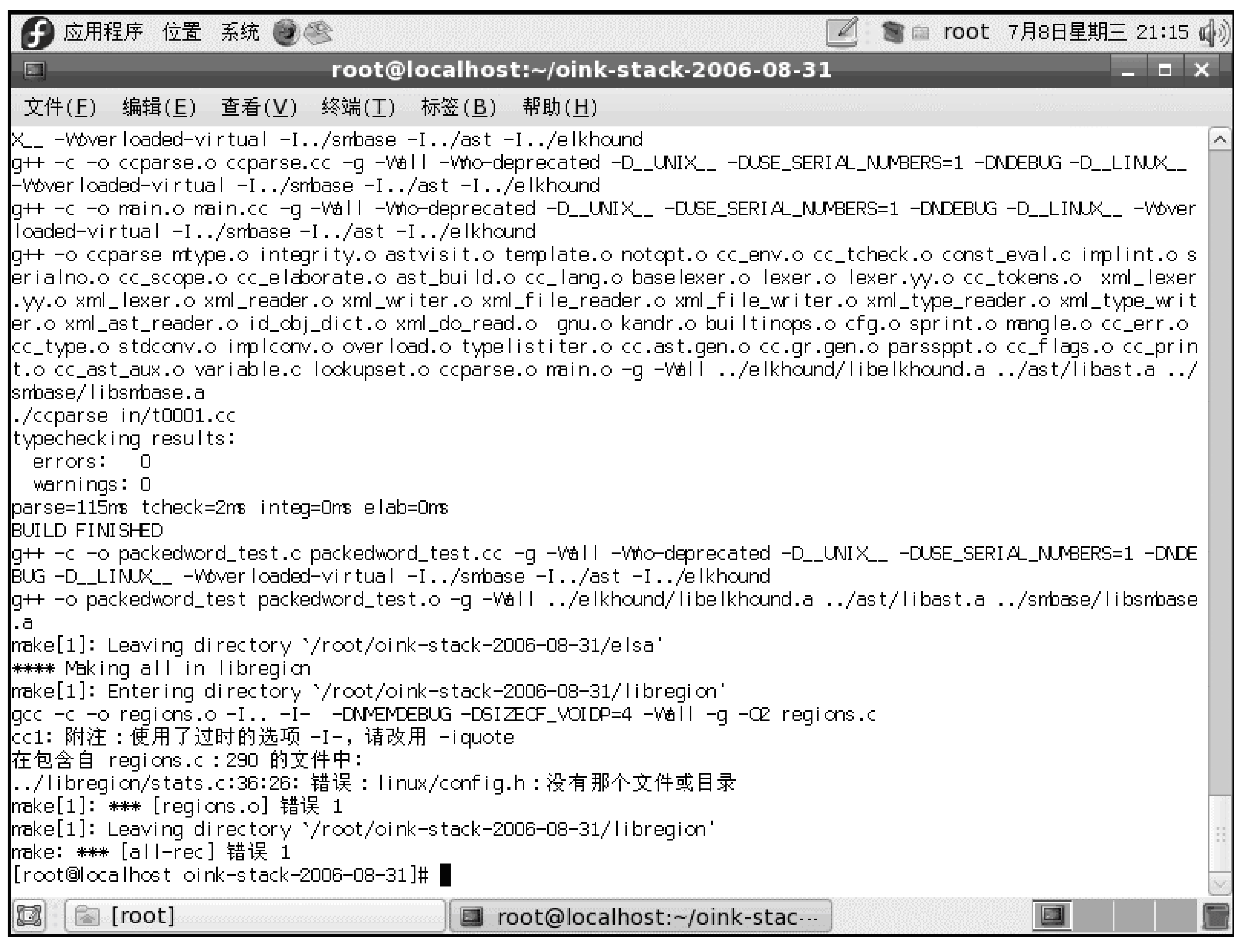


图 3-1　最初安装时可能出现的报错信息

这是由于 Oink 软件与该操作系统的内核版本存在兼容性问题。根据提示，将 libregion 文件夹里 stats.c 文件的第 36 行的#include<linux/config.h>去掉或者注释掉，问题即可解决。

再重新运行：./configure && make clean all check。

再经过较长时间的等待，如果出现如图 3-2 所示的结果，则说明安装成功。

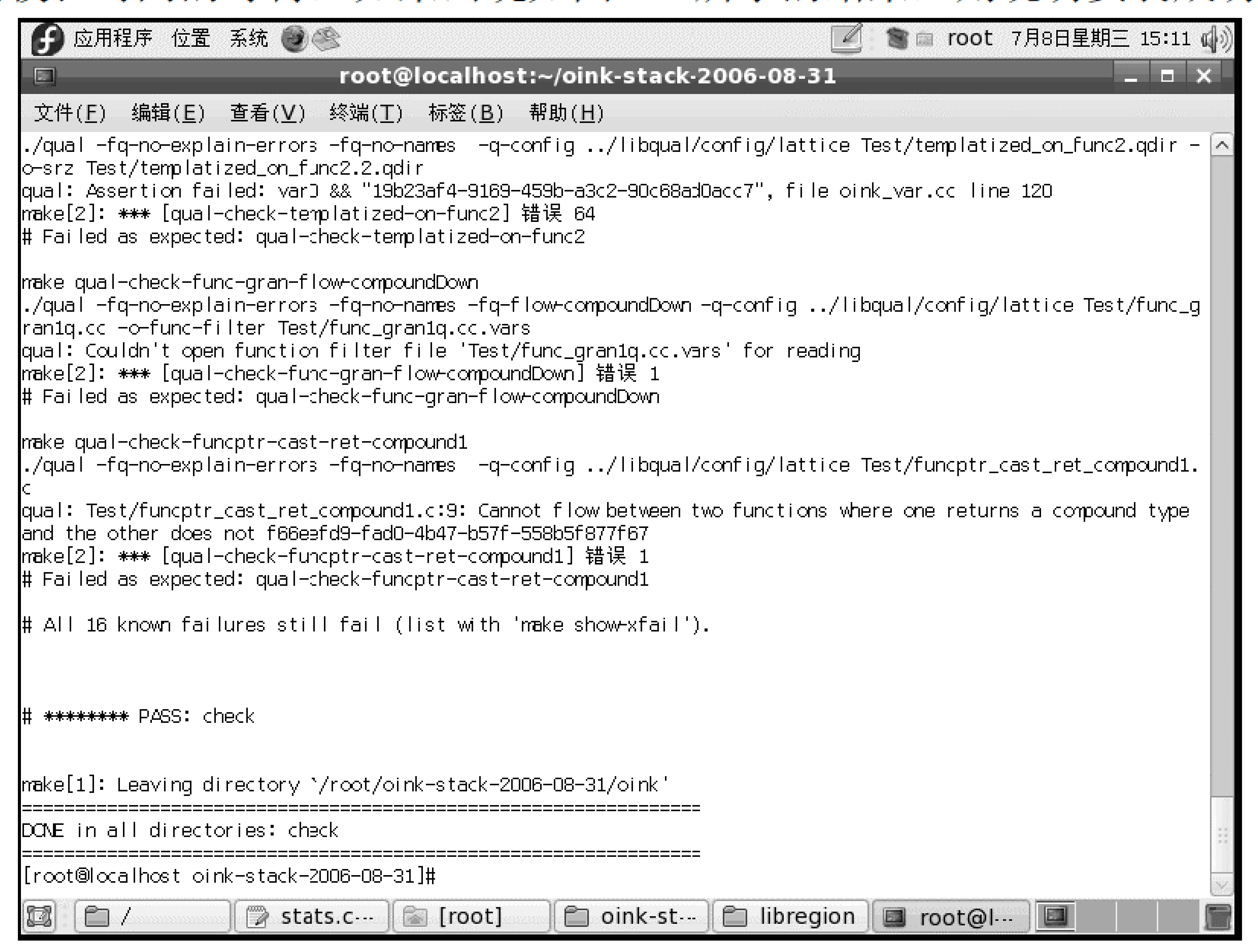


图 3-2　Oink 安装成功

Windows 系统下 Oink 环境的建立步骤如下。

(1) 安装 cygwin 软件。可以在网站 http://www.cygwin.com/上找到 cygwin 的下载和安装信息，这里不再详述。

(2) 打开 cygwin，将下载的 Oink 压缩包在 cygwin 下解压缩。

(3) 在 cygwin 下进入刚才解压缩后的文件夹，运行“./configure && make clean all check”。当出现“DONE in all directories: check”提示时，说明安装成功。

安装时需注意以下几点。

(1) 在安装之前最好查看系统的 GCC 版本，以及安装 Oink 所依赖的 Flex、Bison、Python、Perl 等软件是否安装，并且版本是否合适。GCC 版本可以是 gcc 3.2、gcc 3.4、gcc 4.0、gcc 4.1。有些最新的 Linux 系统默认安装的 gcc 版本比较高(如 gcc 4.3)，安装时会提示有许多 C++源程序语法错误，如找不到一些标准库文件、命名空间定义不对等。要安装成功，就需要手动将这些问题在源代码中按标准 C++规则修改过来。Flex 版本可以是 flex 2.5.4、flex 2.5.31、flex 2.5.33。Bison 版本可以是 bison 1.35、bison 2.1、bison 2.3。Python 版本可以是 python 2.4。即使 GCC 版本适合，Perl 也最好是 perl 5.8.8 以上的版本。即使都满足以上要求，也有可能无法直接安装成功，这主要是由于 Oink 与系统内核存在兼容性问题，这些问题可以通过手动修改代码来解决。

(2) 可以根据具体的应用，安装可选的依赖软件 dot、libzipios++和 zlib。例如，为了将 Oink 的数据流、控制流等分析结果以图形化形式显示，就需要用到 dot 工具，dot 工具可以将分析生成的 dot 格式文件转换为*.ps 格式的图形文件。dot 工具可以从网站 http://www.graphviz.org/下载，上面会有详细的安装说明。libzipios++和 zlib 允许 Oink 读取和生成许多*.qz 和*.qdir 格式的存档文件。

3.2.2　Oink 工具及使用流程

Oink 工具主要包括以下几个。

1. oink

该工具不做任何程序分析工作，它与其他的工具如 staticprint、cfgprint、dfgprint 等共享命令行标志，这几个工具则在此基础上增加一些各自不同的命令行标志。当在终端输入“./oink –help”后，将输出以下结果，这些信息说明了各个命令参数的作用：

```
All arguments not starting with a '-' are considered to be input files.
oink flags that take an argument:
  -o-lang LANG                : specify the input language; one of:
        KandR_C, ANSI_C89, ANSI_C99, GNU_C, GNU_KandR_C, GNU2_KandR_C,
        ANSI_Cplusplus, GNU_Cplusplus, SUFFIX.
  -o-program-files FILE       : add *contents* of FILE to list of input files
  -o-control FILE             : give a file for controlling the behavior of oink
  -o-func-filter FILE         : give a file listing Variables to be filtered out
  -o-srz FILE                 : serialize to FILE
```

```
oink boolean flags; precede by '-fo-no-' for the negative sense.
  -fo-help, -help, --help       : print this message and exit
  -fo-verbose                   : print the setting of each flag
  -fo-print-stages              : announce each processing stage
  -fo-exit-after-parse          : exit after parsing
  -fo-exit-after-typecheck      : exit after typechecking
  -fo-exit-after-elaborate      : exit after elaborating
  -fo-print-startstop           : delimit transformed output with cut lines
  -fo-func-gran                 : compute and print function granularity CFG only
                                  (use -o-srz to write to file)
  -fo-func-gran-dot             : print function granularity CFG in dot format
  -fo-all-pass-filter           : assert that all variables pass the filter
  -fo-print-ast                 : print the ast
  -fo-print-typed-ast           : print the ast after typechecking
  -fo-print-elaborated-ast      : print the ast after elaboration
  -fo-print-ML-types            : print types in ML-style; AST print, not pretty-pr
  -fo-print-buckets             : print buckets
  -fo-print-stats               : print analysis stats
  -fo-print-sizes               : print internal data structure sizes and exit
  -fo-print-proc-stats          : print process stats
  -fo-pretty-print              : print the ast as source
  -fo-trace-link                : trace linking
  -fo-report-link-errors        : print un-satisfied/over-satisfied function symbols in linker
  -fo-report-unused-controls    : print controls that go unused
  -fo-print-controls-and-exit   : print the controls and stop
  -fo-do-overload               : do overload res., overriding language default
  -fo-do-op-overload            : do op overload res., overriding language default
  -fo-do-elaboration            : do elaboration (for C++)
  -fo-check-AST-integrity       : check the AST is a tree
  -fo-exclude-extra-star-amp    : exclude consecutive '&*' or '&*' exprs
  -fo-merge-E_variable-and-var-values :
            optimization: merge Values for E_variable expressions and Variables
  -fo-instance-sensitive        : C mode only.
                                      1) fields get unique values per struct instance
                                      2) 'void's get unique values by type
  -fo-array-index-flows         : the array index flows through the array deref
```

2. staticprint

这是一个静态输出工具，可以输出程序的许多静态分析结果。比如，可以输出类继承关系树图和 AST 节点(histogram of AST nodes)。

在 oink 目录下，输入“./staticprint –help”可查看该工具运行时的详细命令参数信息。

为了方便初学者学习，oink 包里面已经写好了一些运行 staticprint、dfgprint、cfgprint 和 Cqual++工具的 makefile 文件。

在 oink 目录下输入“make staticprint-check-print”，将会运行 staticprint 工具，生成 Test/staticprint1.cc 文件的类继承图。程序代码如图 3-3 所示(生成的图与代码中元素的位置有关，所以这里用代码文件截图来说明它们之间的关系，后面的 dfgprint、dfgprint、cfgprint 和 Cqual++工具也是出于同样的原因，对代码文件进行了截图)。



图 3-3　类继承信息

运行后生成的类继承图如图 3-4 所示(参见彩图)。

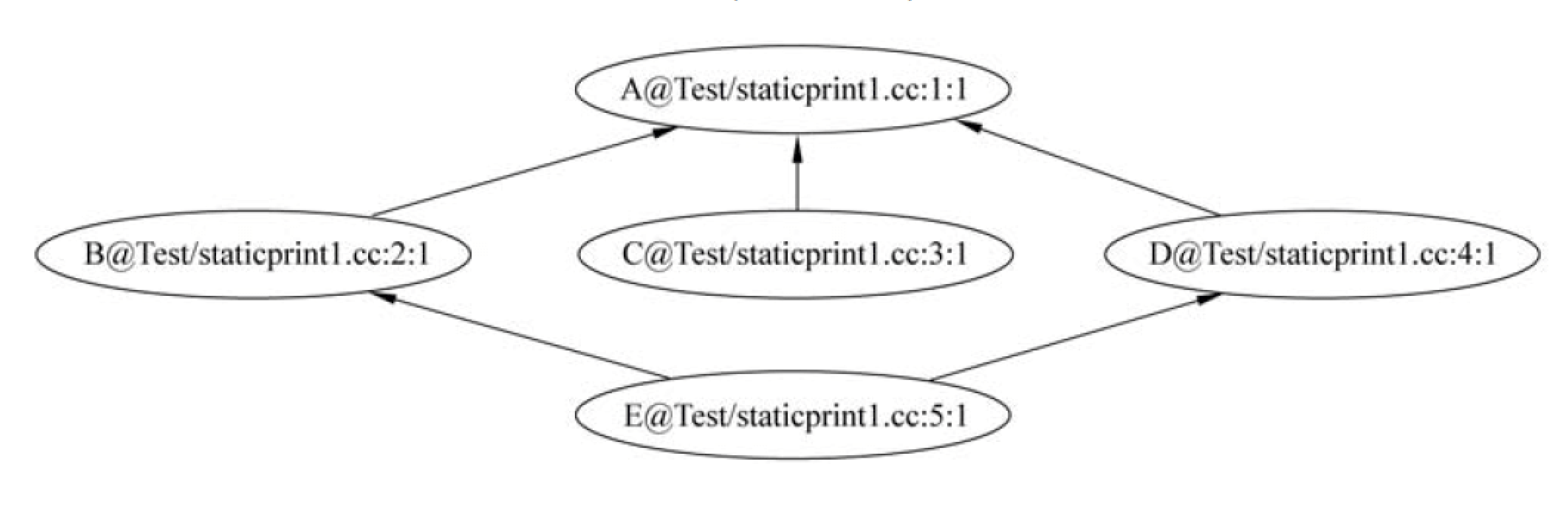


图 3-4　类继承图

从图 3-4 中可以看出 A、B、C、D、E 这 5 个类之间的继承关系，蓝箭头表示的是虚继承，黑箭头表示的是一般继承。

如果要分析其他程序，则可以修改 staticprint_test.incl.mk 文件中的文件名，然后运行即可，流程与上面的相同。

3. dfgprint

用来输出 C++程序的数据流图。

在 oink 目录下输入“make dfgprint-check-print”，将会运行 dfgprint 工具，生成 Test/dfgprint1.c 文件的数据流图。程序代码如图 3-5 所示。

dfgprint1.c - Microsoft Visual Studio

文件(F)　编辑(E)　视图(V)　调试(D)　工具(T)　测试(S)　窗口(W)　帮助(H)

dfgprint1.c

```
int g(int m)
{
 int j;
 int k=0;
 for(j=1;j<m+1;j++)
{
 k+=j;
}
return k;

}
int f(int a,int b) {
int n;
 if(a>=b)
 n=g(a);
 else
 n=g(b);
return a*b;

}

int main() {
   int IsBig=0;
   int x = 3;
   int y= 5;
   int z=f(x,y);
   if(z>100)
  {
   IsBig=1;
  }

}
```

图 3-5　程序示例 1

图 3-6 中(参见彩图)显示了数据流向，红色箭头表示在调用时函数，实参传给形参，数据流向了被调用函数，而蓝色箭头则表示被调用函数将数据返回到调用处的数据流向，至于黑色箭头，则表示一般的数据流向。而对于每个节点来说，@符号之前部分表示数据状态的名称，@符号之后部分表示数据在代码中的位置。如果要分析其他程序，则可以通过修改或添加 dfgprint_test.incl.mk 文件中的 CHK_DFGPRINT 信息来实现。例如：

CHK_DFGPRINT+=Test/dfgprint1.c

运行后生成的数据流图如图 3-6 所示。

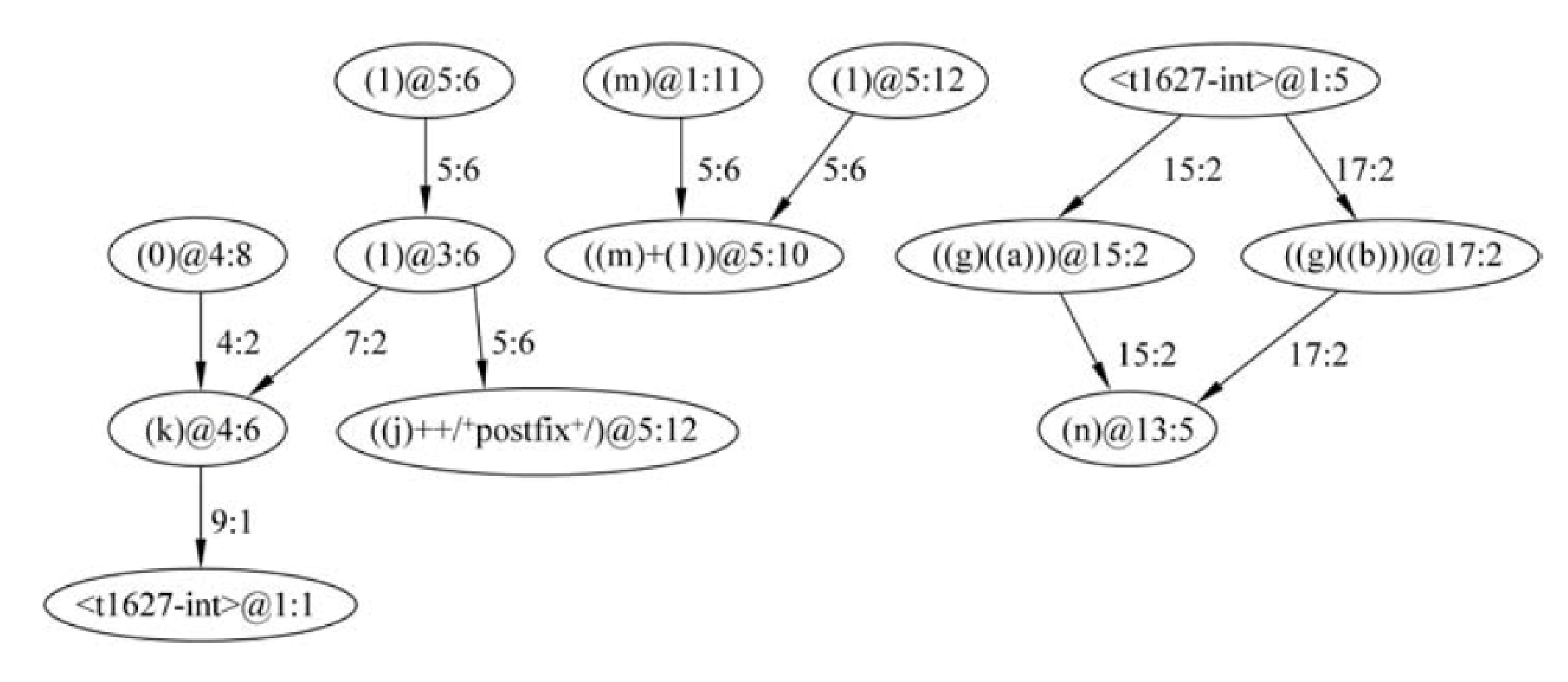


图 3-6　程序示例 1 的数据流图

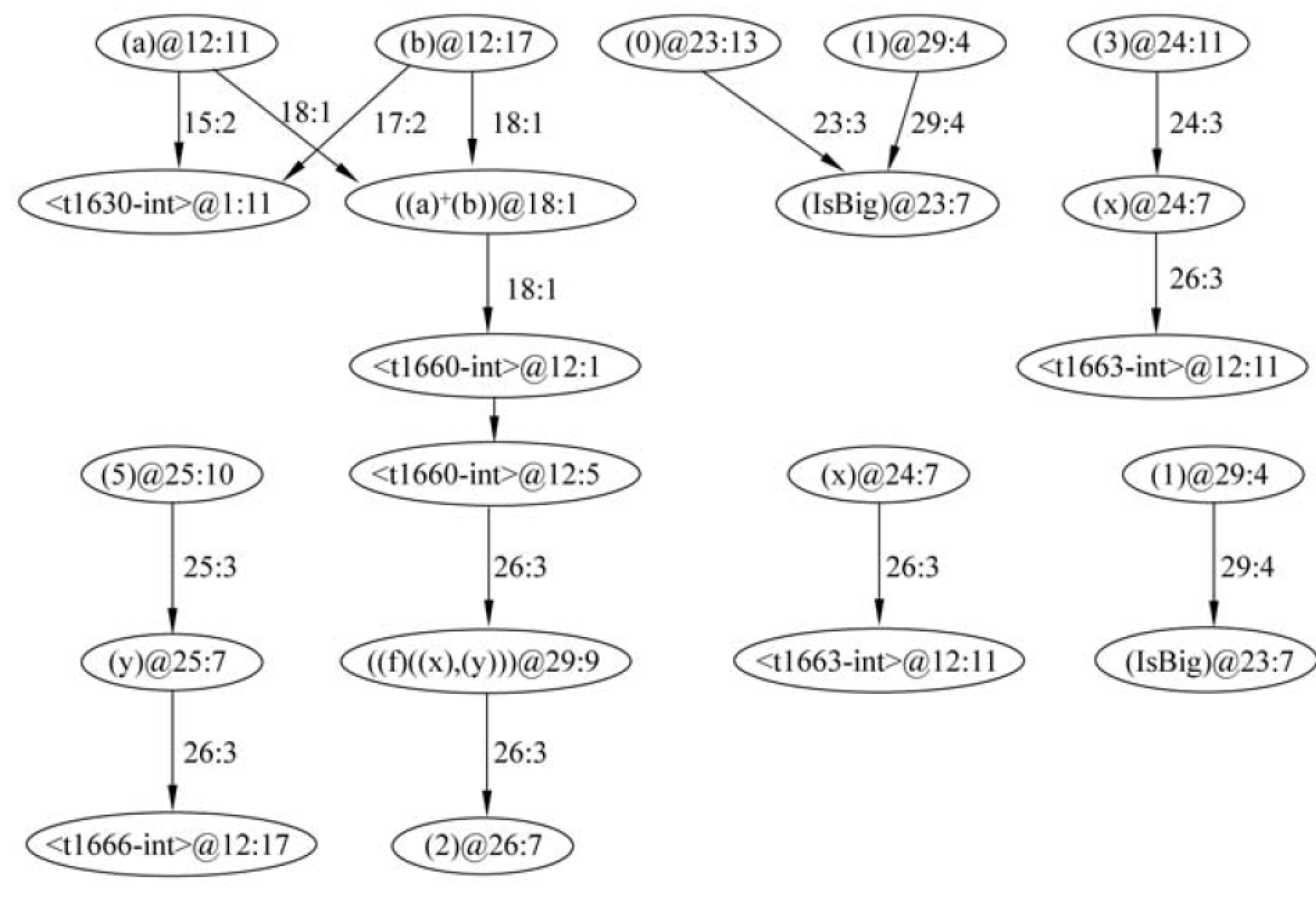


图 3-6　(续)

4. cfgprint

用来输出 C++程序的控制流图。

在 oink 目录下输入“make cfgprint-check-print”，将会运行 cfgprint 工具，生成 Test/cfgprint1.c 文件的数据流图。程序代码如图 3-7 所示。

```
cfgprint1.c

int checkGrade(int g)
{
    int result;
    switch(g)
    {case 1:
            result=1;
            break;
        case 2:
            result=1;
            break;
        case 3:
            result=2;
            break;
        case 4 :
            result=2;
            break;
        default:
            result=3;
            break;
    }
    return result;

}

int main() {

    int m=4;
    int n=0;
       int i;
    for(i=0;i<=m;i++)
    {
         n+=i;
    }
    return  (checkGrade(n/m));

 }
```

图 3-7　程序示例 2

运行后生成的控制流图如图 3-8 所示。

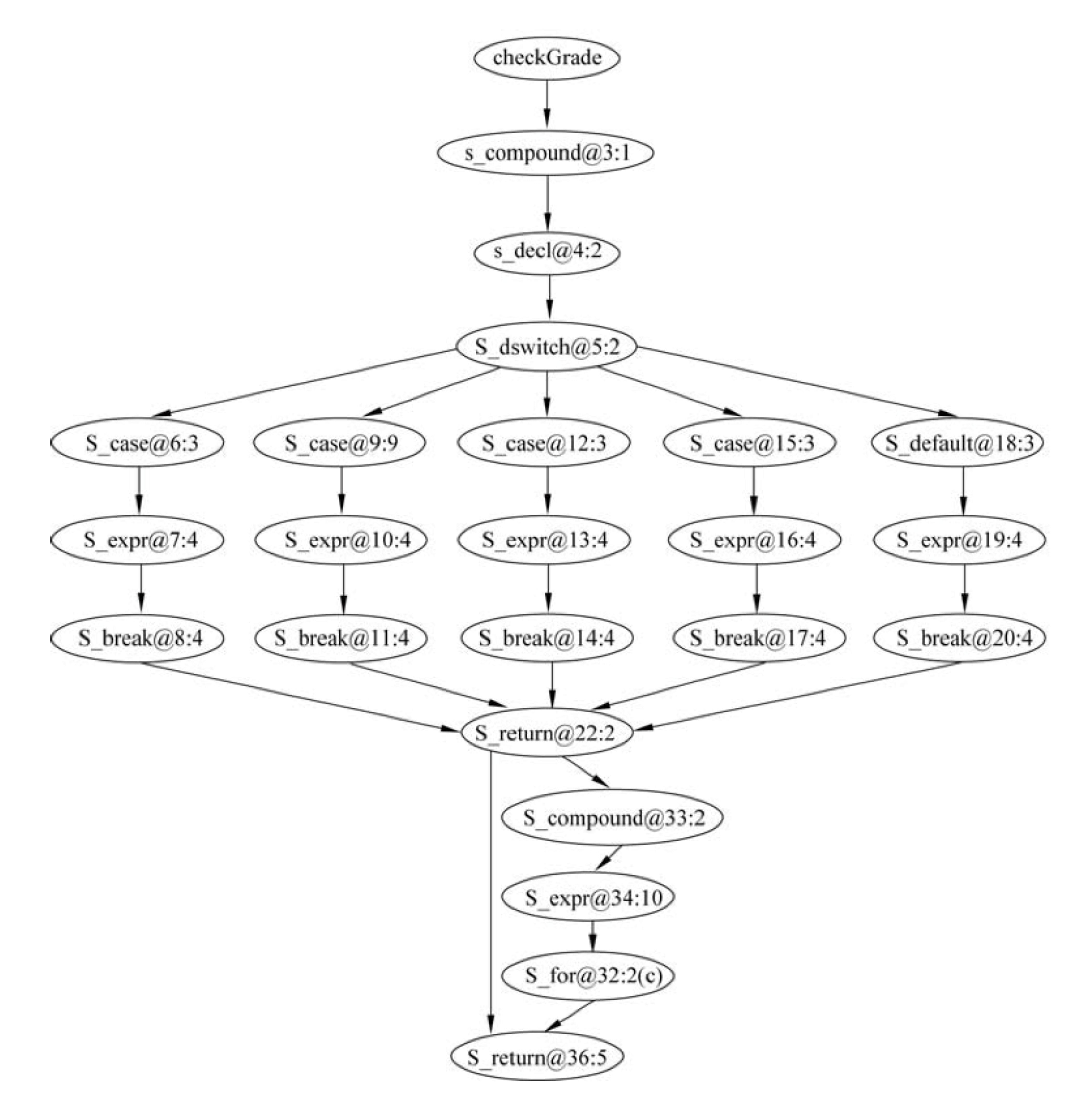


图 3-8　程序示例 2 的控制流图

图 3-8 中显示了程序的控制结构，如果要分析其他程序，则可以修改 cfgprint_test.incl.mk 文件中的文件名，然后运行，具体流程与上面的相同。

5. Cqual++

Cqual++是 Oink 的主要工具，它有许多功能，可以通过检验 C++程序的静态断言来对程序进行分析和理解。在 oink 目录下输入“./qual –help”，可以查看 Cqual++工具的详细命令行参数。

3.2.3　Oink 应用举例

目前，Oink 的主要用途是对代码进行数据流和控制流的分析以及基于断言的类型修饰符分析。数据流和控制流分析的使用方法在 3.2.2 节已经说明，而基于断言的类型修饰符分析可以对代码进行格式字符串缺陷(format string bug)分析。

下面是一个 printf()函数的格式字符串缺陷的例子，如图 3-9 所示为 Test/taint1.c 程序的代码。

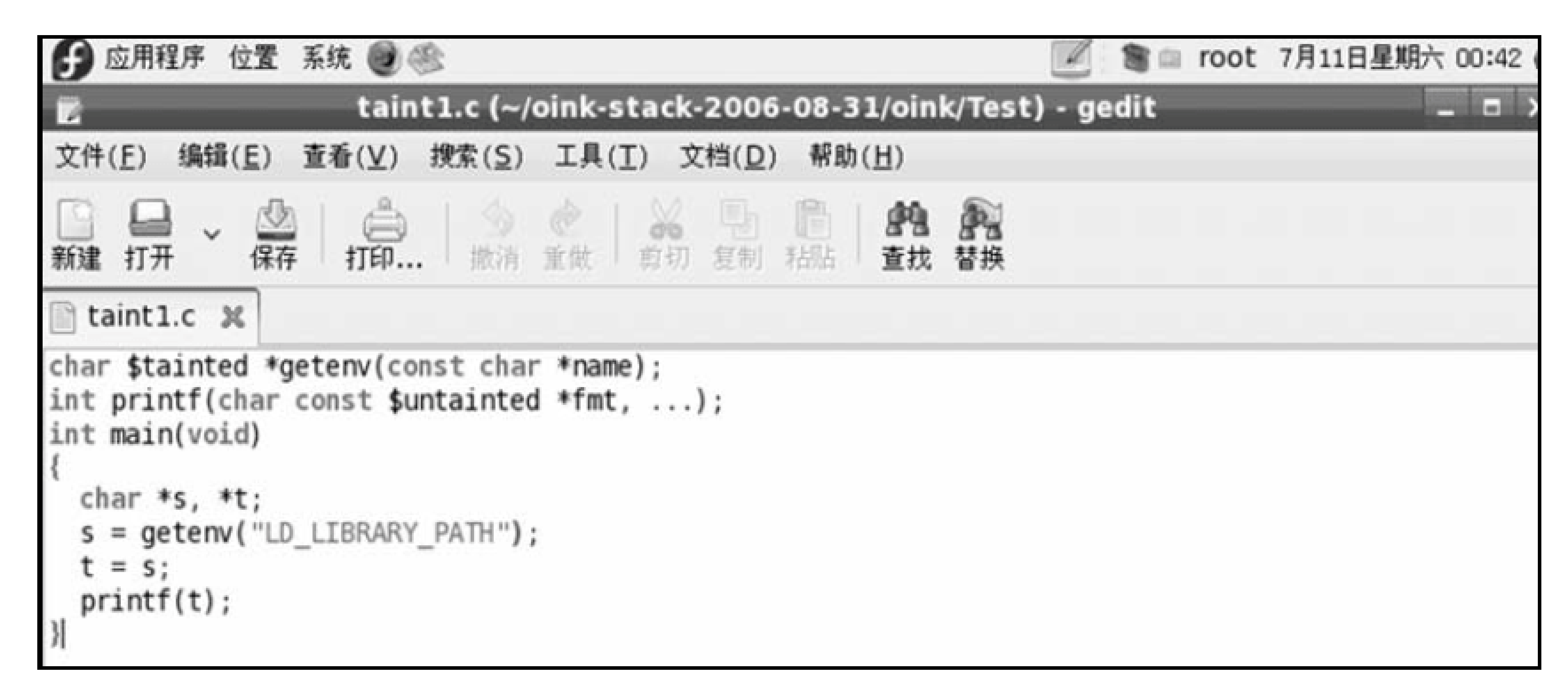


图 3-9　程序示例 3

图 3-9 中多了$tainted 和$untainted 这两个在 C 语言语法中没有的修饰符，它们是 Oink 添加的类型修饰符(如果去掉它们，用 GCC 编译器编译运行，将会发现并没有提示错误，这说明 GCC 没有发现程序有错误)。分析程序可知，这个程序要通过函数 getenv()读取环境变量 LD_LIBRARY_PATH，并将这个变量以格式字符串形式传递给 printf()函数。如果一个不能信任的用户在程序运行时随便更改了这个环境变量，那么程序将可能出现格式字符串漏洞。比如，如果用户将环境变量 LD_LIBRARY_PATH 设置为一个很长的字符串，那么可能超出内存空间，产生因越界而使程序终止的段错误。为了查找这样的错误，Oink 在程序中添加了两个类型修饰符——$tainted 和$untainted。$tainted 表示不可相信的数据，而$untainted 则表示可以相信的数据。在第一行定义函数 getenv()的返回值是不可以相信的($tainted)，通过该函数可以得到用户设置的任何环境变量数据；第二行定义的函数 printf()的形参表中的参数是可以相信的($untainted)。

图 3-10 是运行“make qual-check-demo-taint1”后的结果。

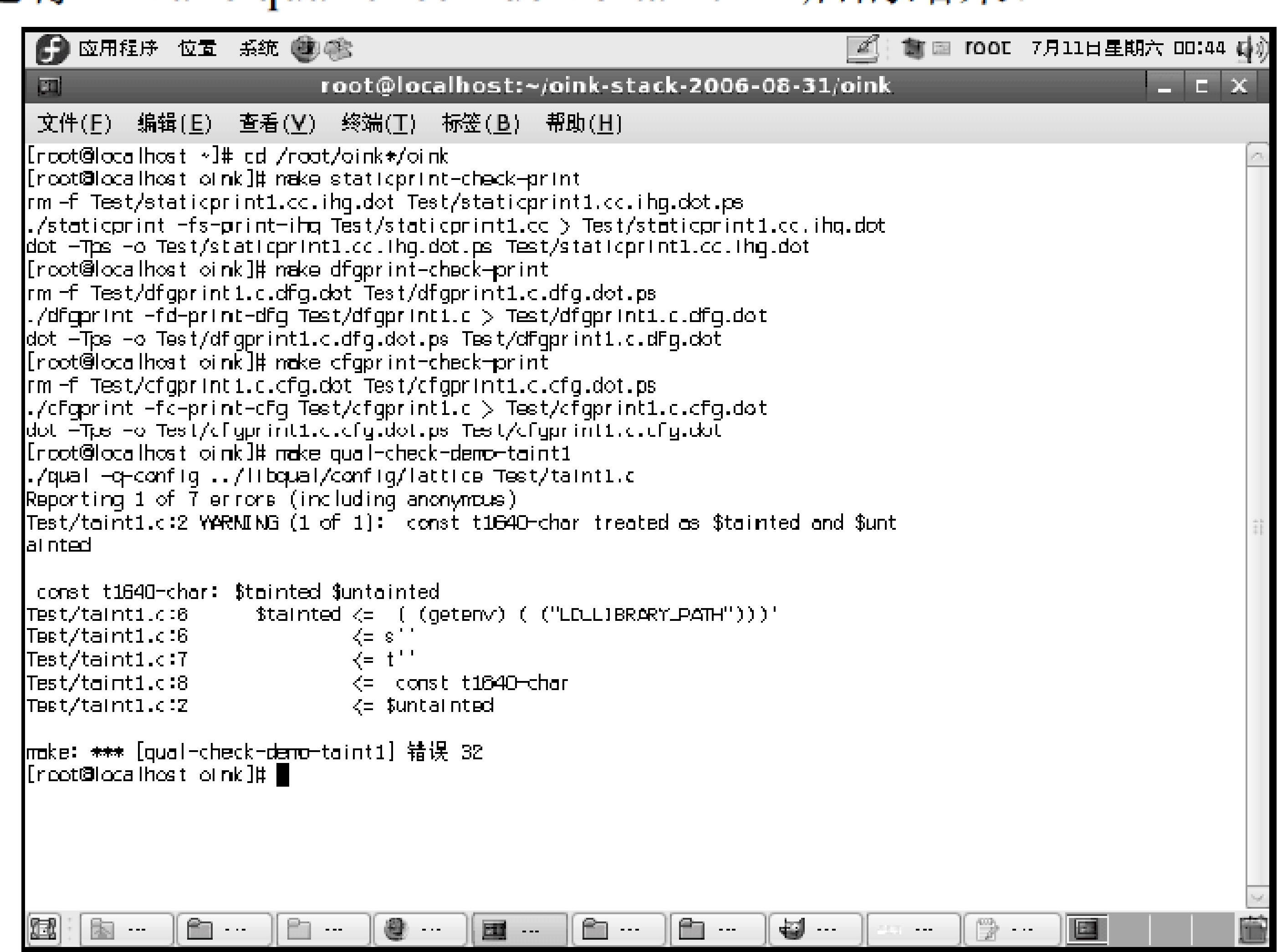


图 3-10　程序示例 3 的分析结果

由图 3-10 可以看出，程序运行结果的第 1 行，即“const t1640-char: $tainted $untainted”

表示的意思是 t1640-char，也就是示例 3Test/taint1.c 中的变量 t 既是$tainted 类型，又是$untainted 类型。前面讲过，$tainted 和 $untainted 是 Oink 增添的两个数据类型，分别表示不安全数据和安全数据。那么在程序运行结果中得到了 t1640-char 既是安全的又是不安全的，这明显是矛盾的，我们认为是一个错误。从程序运行结果的第 2 行到最后一行给出了产生错误的根本原因：在程序中定义 getenv(const char * name)的返回类型是$tainted，然后赋给变量 s，又赋给变量 t，最后输出打印。而对于输出打印的数据，则要求是$untainted 类型，这样就得出了$tainted<=$untainted。那么在数据流中从不安全的数据传向安全的数据，就使得安全的数据也变得不安全了，这是极其危险的操作。

Oink 除了可以进行上面的简单分析外，还可以对多态类型提供支持。例如如图 3-11 所示的程序。

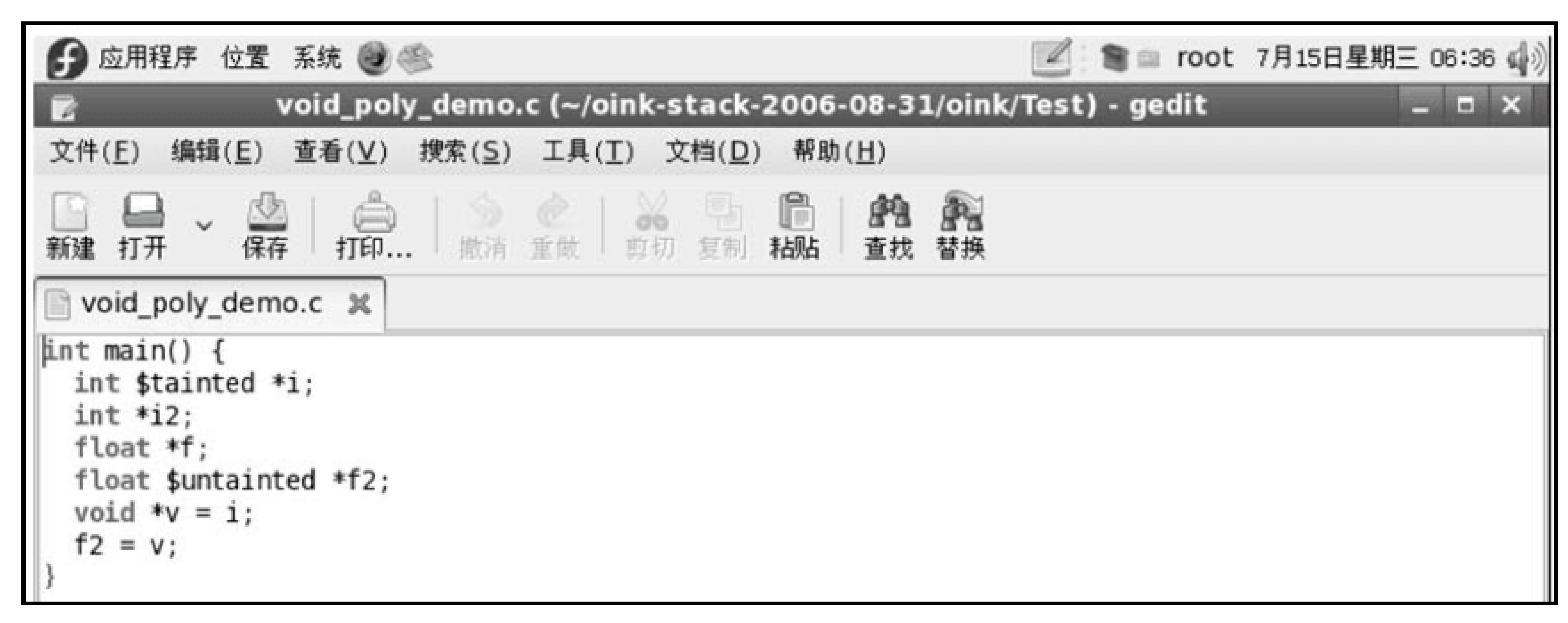


图 3-11　程序示例 4

图 3-11 的代码先将 int 型指针 i 赋给 void 型指针 v，再将 void 型指针 v 赋给 float 型指针 f2。运行“make qual-check-demo-void-poly”之后将会出现如图 3-12 所示的运行结果。

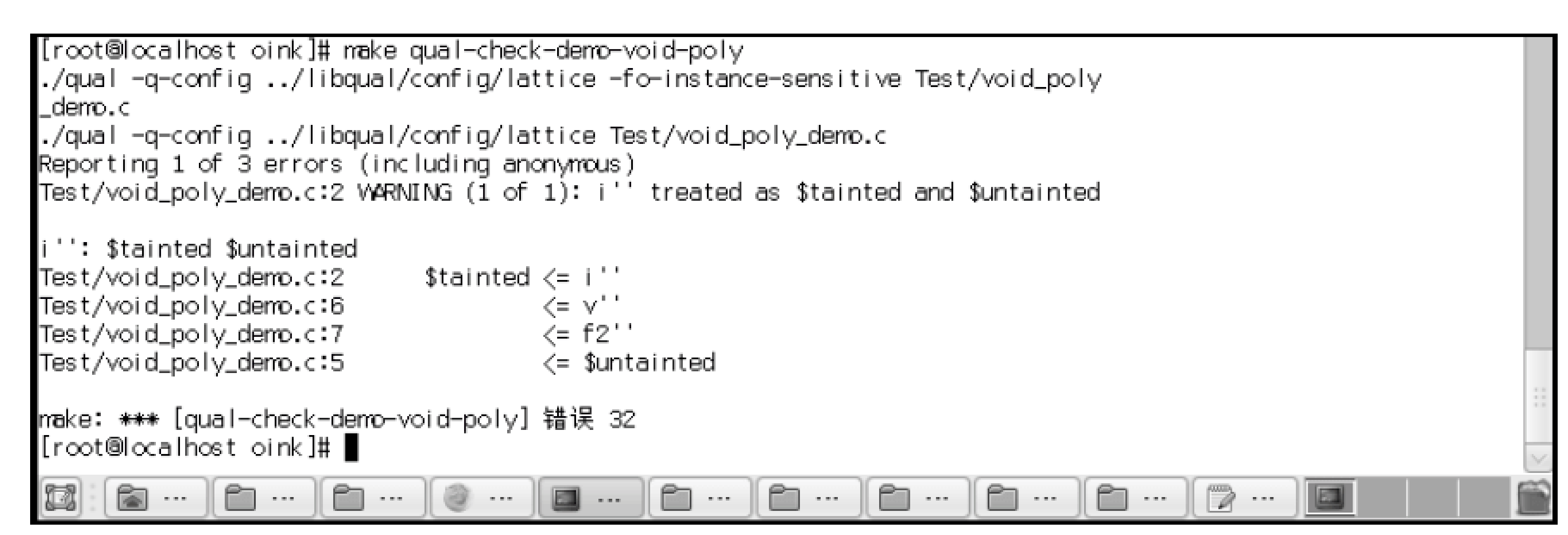


图 3-12　程序示例 4 的分析结果

由图 3-12 可以看出，程序运行结果的第 1 行，即“i’’:$tainted $untainted”发生错误。这个错误和示例 3 中的错误是一样的，都得出了一个变量既是安全的又是不安全的结果。我们知道在示例 4 中，定义 i 为$tainted 类型、f2 为$untainted 类型。然后把 i 赋给 v，又把 v 赋给 f2，那么数据流向就从不安全数据流向了安全数据，从而导致安全数据变得不安全了，这同样是数据流中极其危险的操作。

Oink 除了可以进行格式字符串缺陷分析之外，还可以找出代码中的 user-kernel 缺陷。内核必须保证用户代码中的指针所指的内存是合法的，但不允许用户代码操纵内核数据，否则代码也将是不安全的。Oink 的开发者们使用 Oink 对 Linux 内核源码进行了 user-kernel

分析，从 2.4.20 版的内核中找到 7 个 bug 和 275 个 false pos，从 2.4.23 版的内核中找到 6 个 bug 和 264 个 false pos。

3.3 Eclipse PTP/CDT 程序理解工具

Eclipse CDT 是 Eclipse 插件，它把 Eclipse 转换为功能强大的 C/C++ IDE。它被设计为将 Java 开发人员喜爱的许多 Eclipse 优秀功能提供给 C/C++开发人员，例如项目管理、集成调试、类向导、自动构建、语法着色和代码完成等。当 Eclipse 被用作 Java IDE 时，它将利用 JDK 并与之集成。同样地，CDT 将利用标准的 C/C++工具并与之集成，例如 g++、make 和 GDB，这使得 CDT 在 Linux 中变得非常流行。这些工具都可在 Linux 中使用并用于大多数 C++开发。可以在 CDT 的基础上安装 PTP，以便通过 PTP 进行更好的静态分析。

3.3.1 PTP/CDT 介绍

1. CDT 介绍

CDT 是完全用 Java 实现的开源项目(根据 Common Public License 特许的)，它作为 Eclipse SDK 平台的一组插件。这些插件将 C/C++透视图添加到 Eclipse 工作台(Workbench)中，现在后者可以用许多视图和向导以及高级编辑和调试支持，来支持 C/C++开发。

由于 CDT 的复杂性，CDT 被分成几个组件，它们均采用独立插件的形式。每个组件都作为一个独立自主的项目进行运作，有它自己的一组提交者、错误类别和邮件列表。但是，所有插件都是 CDT 正常工作所必需的。下面是 CDT 插件/组件的完整列表。

(1) 主 CDT 插件(Primary CDT plug-in)：是“框架”CDT 插件。

(2) CDT 功能 Eclipse(CDT Feature Eclipse)：是 CDT 功能组件(Feature Component)。

(3) CDT 核心(CDT Core)：提供了核心模型(Core Model)、CDOM 和核心组件(Core Component)。

(4) CDT UI：是核心 UI、视图、编辑器和向导。

(5) CDT 启动(CDT Launch)：为诸如编译器和调试器之类的外部工具提供了启动机制。

(6) CDT 调试核心(CDT Debug Core)：提供了调试功能。

(7) CDT 调试 UI(CDT Debug UI)：为 CDT 调试编辑器、视图和向导提供了用户界面。

(8) CDT 调试 MI(CDT Debug MI)：是用于与 MI 兼容的调试器的应用程序连接器。

CDT 的开发有两个主要目标：一是在 Eclipse 上提供一个用来进行 C/C++开发的平台；二是提供支持 GNU 工具链的 API，例如支持 GNU make-based build systems、GCC/G++ compilers、GDB debugger 等。

其中用来进行源代码静态分析的信息存储在 CDT 核心之一——CDOM 中。CDOM 中有一个文件对象模块，在这个模块中会产生 DOM AST(Abstract Syntax Tree)，所有进行静态分析的信息均是通过 AST 来产生的，如图 3-13 所示。

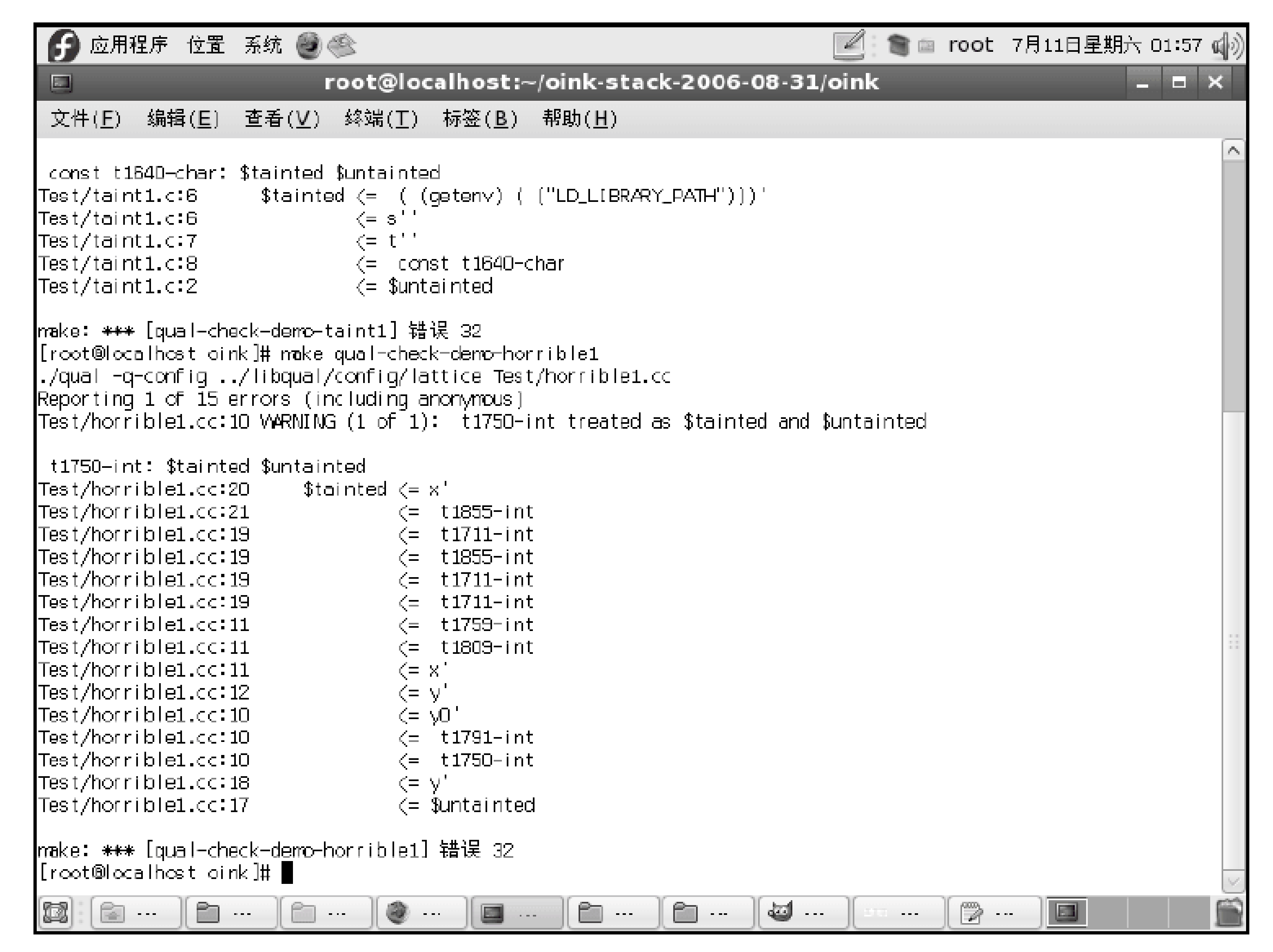


图 3-13　AST 信息输出

2. PTP 介绍

PTP(Parallel Tools Platform)作为 Eclipse 的插件，其开发目的是为了生产出一个开源的工业化的平台，特别是为并行应用开发提供一个高度整合的环境。PTP 插件有以下 4 个主要领域。

(1) 运行时的工具，使开发人员能够检测和控制执行并行应用程序。

(2) 调试工具，用于查找运行并行应用程序中的错误。

(3) 分析工具，提供先进的编辑、错误检查，帮助程序员开发并行应用程序。

(4) 性能工具，对并行应用程序进行性能分析和优化。

其中作为分析工具，PTP 在 CDT 的基础上利用抽象语法树进行静态分析，提供了一系列的高级编辑和错误检查功能。这也是用 PTP 来进行静态分析的原因。

3.3.2　PTP 环境建立

目前，PTP 的最新版本是 PTP 7.0.7；官方网址是 http://www.eclipse.org/ptp。在这里，仍用早期的 PTP 2.1 版，读者也可尝试用新版本。

在安装 PTP 之前需要安装以下软件。

(1) Eclipse：这里将使用 Eclipse 的插件 CDT。

(2) Java Runtime Environment：这里要构建 C++应用程序，而且要使用 Eclipse。而 Eclipse 本身是 Java 应用程序，因此也需要安装 Java Runtime Environment(JRE)。在这里使用 Eclipse V3.2，它要求使用 JRE V1.4 或更高版本。如果还需要使用 Eclipse 进行 Java 开发，则需要安装 Java Development Kit(JDK)。

(3) Eclipse C/C++ Development Toolkit(CDT)：PTP 是在 CDT 的基础上进行安装的，因此当然需要安装 CDT。

(4) Cygwin/MinGW：如果要使用 Microsoft Windows，则 Cygwin /MinGW 十分有用，Cygwin/MinGW 在 Windows 中提供了类似 Linux 的环境。

(5) GNU C/C++ Development Tools：CDT 将使用标准的 GNU C/C++工具来编译代码、构建项目和调试应用程序。这些工具包括 GNU Compiler Collection(GCC)for C++ (g++)、make 和 GNU Project Debugger(GDB)。如果读者是使用 Linux 或 Mac OS X 的程序员，则可以将这些工具安装到计算机上。本节包含针对 Windows 设置这些工具的说明。

PTP 既可以安装在 Windows 上，也可以安装在 Linux 上，下面分两种方式来介绍 PTP 环境的建立。

表 3-1 是安装 PTP 的系统配置要求列表(PTP 版本：2.1)。

表 3-1　PTP 安装要求列表

Component	OS(Eclipse)	OS(Server)	Java	Eclipse	CDT	RSE	MPI	Others
PTP	Linux	Linux	1.5 或更高版本	3.4.1	5.0.1 或更高版本	3.0	Open MPI 1.2.x 或 1.3.x	GDB 6.3～6.8
	Mac OS X	Mac OS X					MPI CH2.1.0.6p1	
	Windows	UNIX					IBM 并行环境	

1. Linux 下 PTP 环境的建立

在 Linux 下安装 PTP 有以下几个基本步骤。

1) Install Java

首先下载 jre-1_5_0_09-linux-i586-rpm.bin。

(1) 新建一个 Java 的目录：

```
[root@localhost~]#mkdir /usr/local/java
```

(2) 将下载的 jre-1_5_0_09-linux-i586-rpm.bin 放到/usr/local/java 目录下。

(3) 进入超级用户模式：

```
[root@localhost~]#su
```

(4) 进入 Java 目录：

```
[root@localhost ~]#cd /usr/java
```

(5) 更改 jre-1_5_0_09-linux-i586-rpm.bin 的权限为可执行：

```
[root@localhost java]#chmod a+x jre-1_5_0_09-linux-i586-rpm.bin
```

(6) 启动安装过程：

```
[root@localhost java]#./jre-1_5_0_09-linux-i586-rpm.bin
```

(此时将显示二进制许可协议，按空格键显示下一页，阅读完许可协议后，输入“yes”继续安装,如图 3-14 所示。此时会将 JRE 解压缩,产生 jre-1_5_0_9-linux-i586.rpm,如图 3-15 所示。)

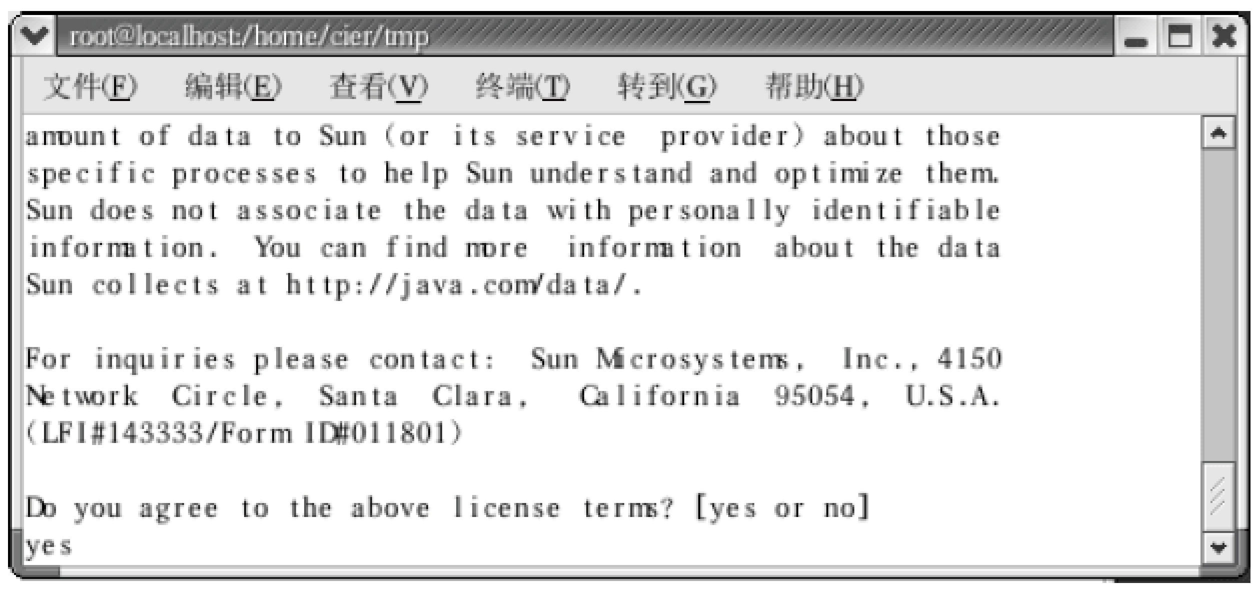


图 3-14　阅读 Sun 有关协议

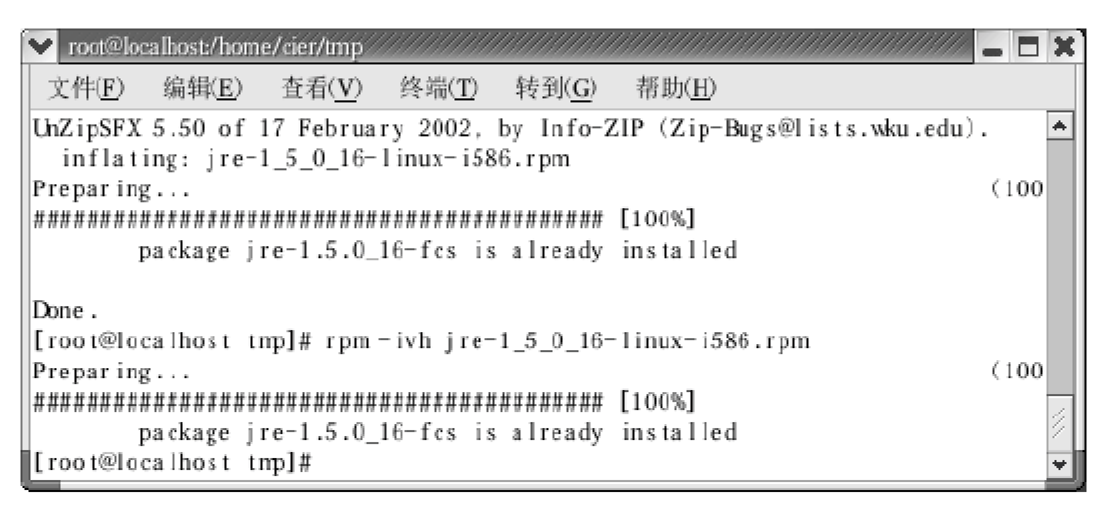


图 3-15　解压缩 JRE

(7) 安装 jre-1_5_0_09-linux-i586-rpm：

[root@localhost java]#rpm –ivh jre-1_5_0_9-linux-i586.rpm

(此时会将 JRE 安装在/usr/java/jre1_1_5_0_09 目录下。)

(8) 设定环境变量，让 Linux 能找到 JRE：

[root@localhost java]#vi /etc/profile

将以下内容添加到文件后面：

```
PATH=$PATH:/usr/java/jre1.5.0_09/bin
export JAVA_HOME=/usr/java/jre1.5.0_09
export CLASSPATH=$JAVA_HOME/lib:.
```

(存盘后，重新启动 Linux。)

(9) 测试 Java 是否安装成功，如图 3-16 所示。

[root@localhost ~]#java –version

2) Install Eclipse

(1) 首先下载 eclipse-java-ganymede-SR1-linux-gtk.tar，可以将其放到桌面上。

(2) 进入要存放 Eclipse 的目录：

[root@localhost~]#cd /usr/loca

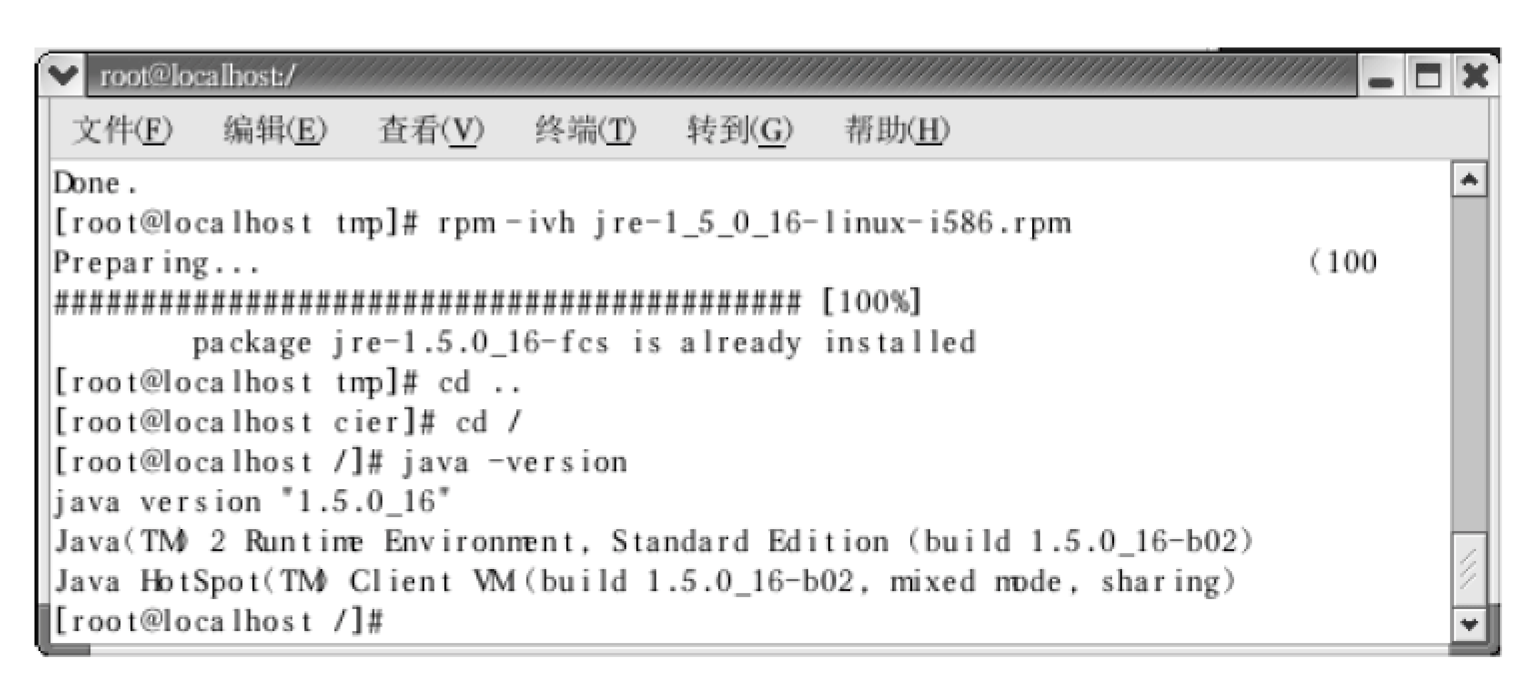


图 3-16　查看 Java 版本信息

(3) 复制 eclipse-java-ganymede-SR1-linux-gtk.tar 到当前目录：

[root@localhost~]#cp ~Desktop/eclipse-java-ganymede-SR1-linux-gtk.tar

(4) 解压缩：

[root@localhost~]#tar –zxvf /eclipse-java-ganymede-SR1-linux-gtk.tar

(5) 进入 Eclipse 目录：

[root@localhost~]#cd eclipse

(6) 执行 Eclipse：

[root@localhost~]#./eclipse

(7) 选择 Java 程序的存放目录，如图 3-17 所示。

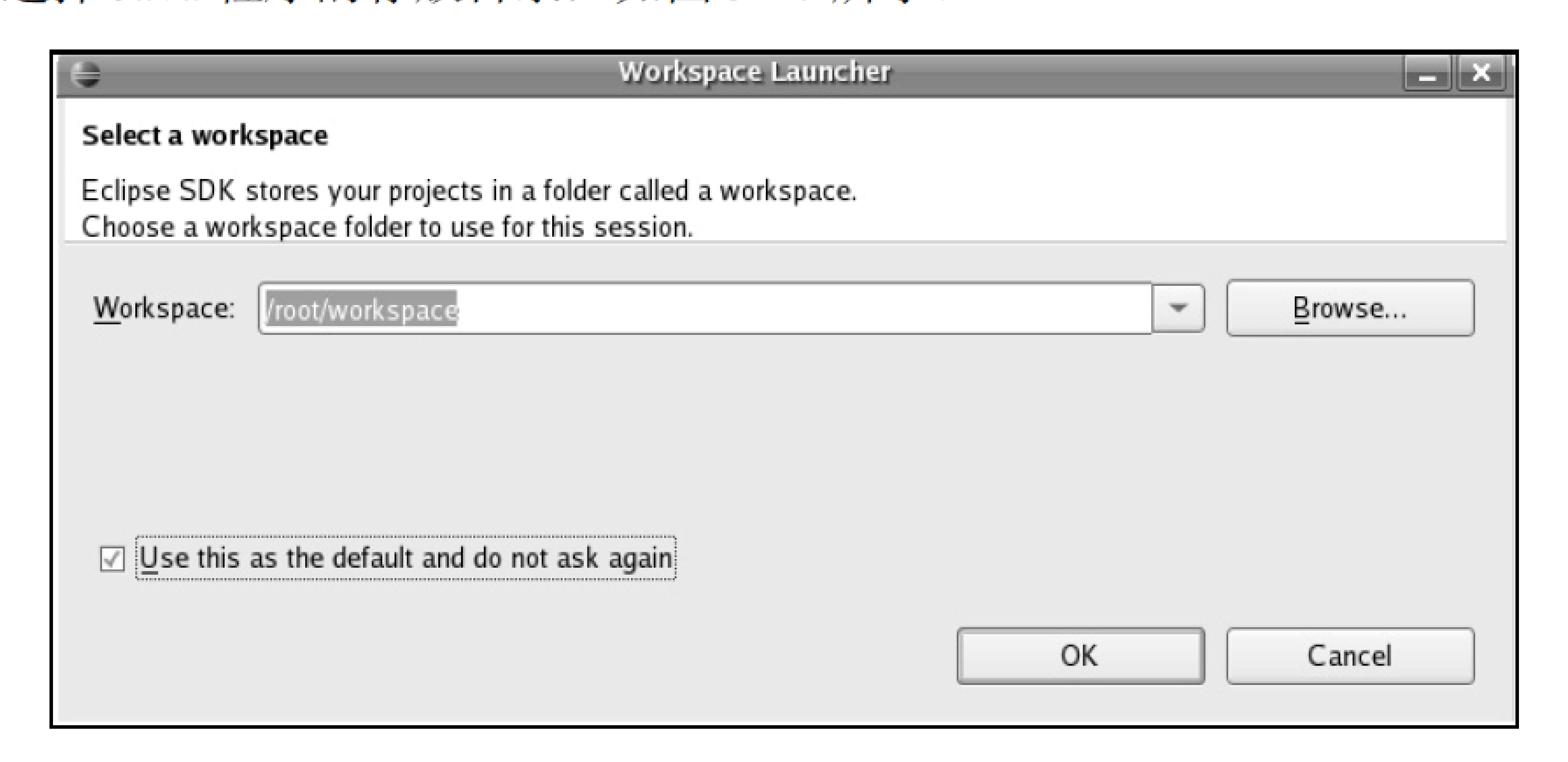


图 3-17　选择 Java 程序的存放目录

(若希望以后不出现此对话框，可选中 Use this as the default and do not ask again 复选框。)

(8) 进入 Eclipse 主界面，如图 3-18 所示。

3) Install CDT

(1) 下载 cdt-master-4.0.1，可将其放到桌面上。

(2) 进入存放 Eclipse 的目录，如图 3-19 所示。

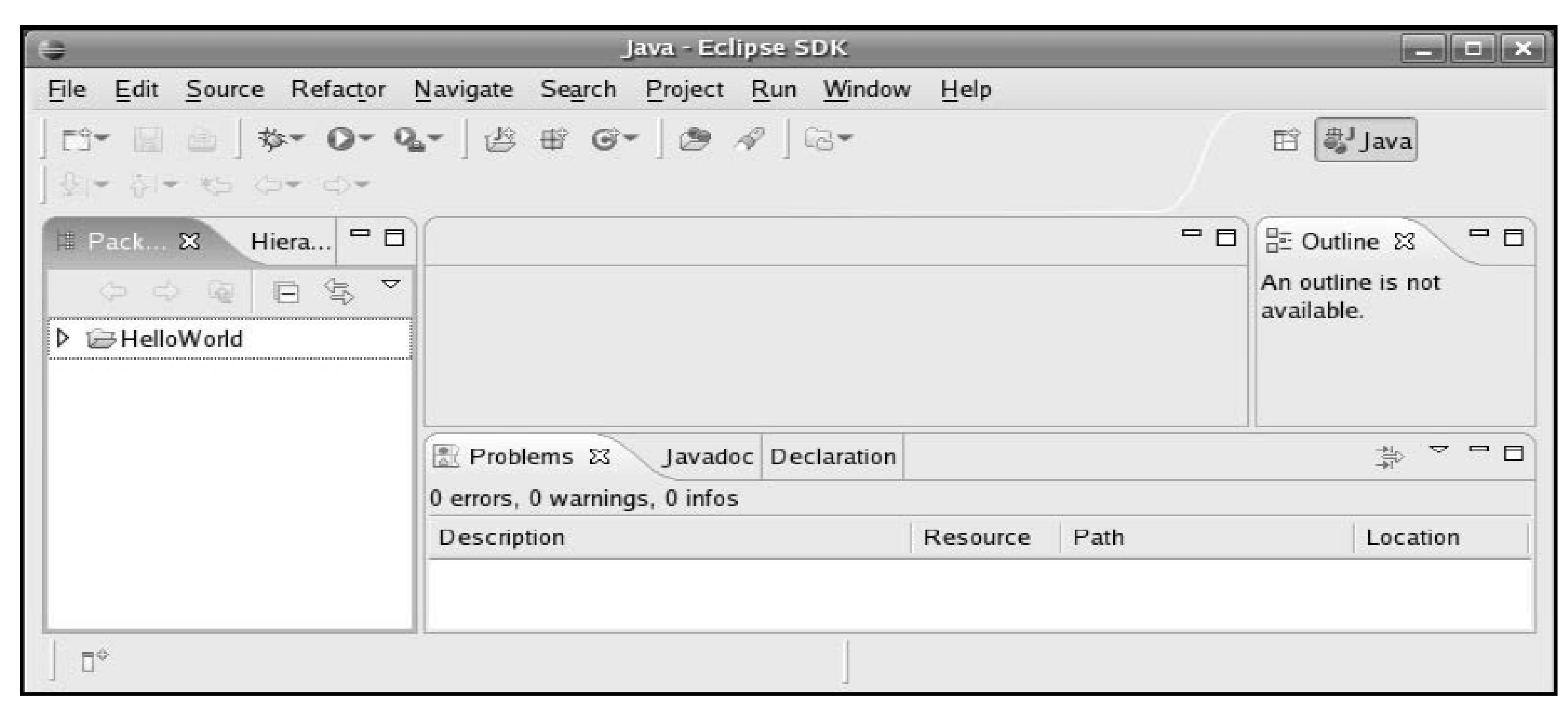


图 3-18　Eclipse 主界面

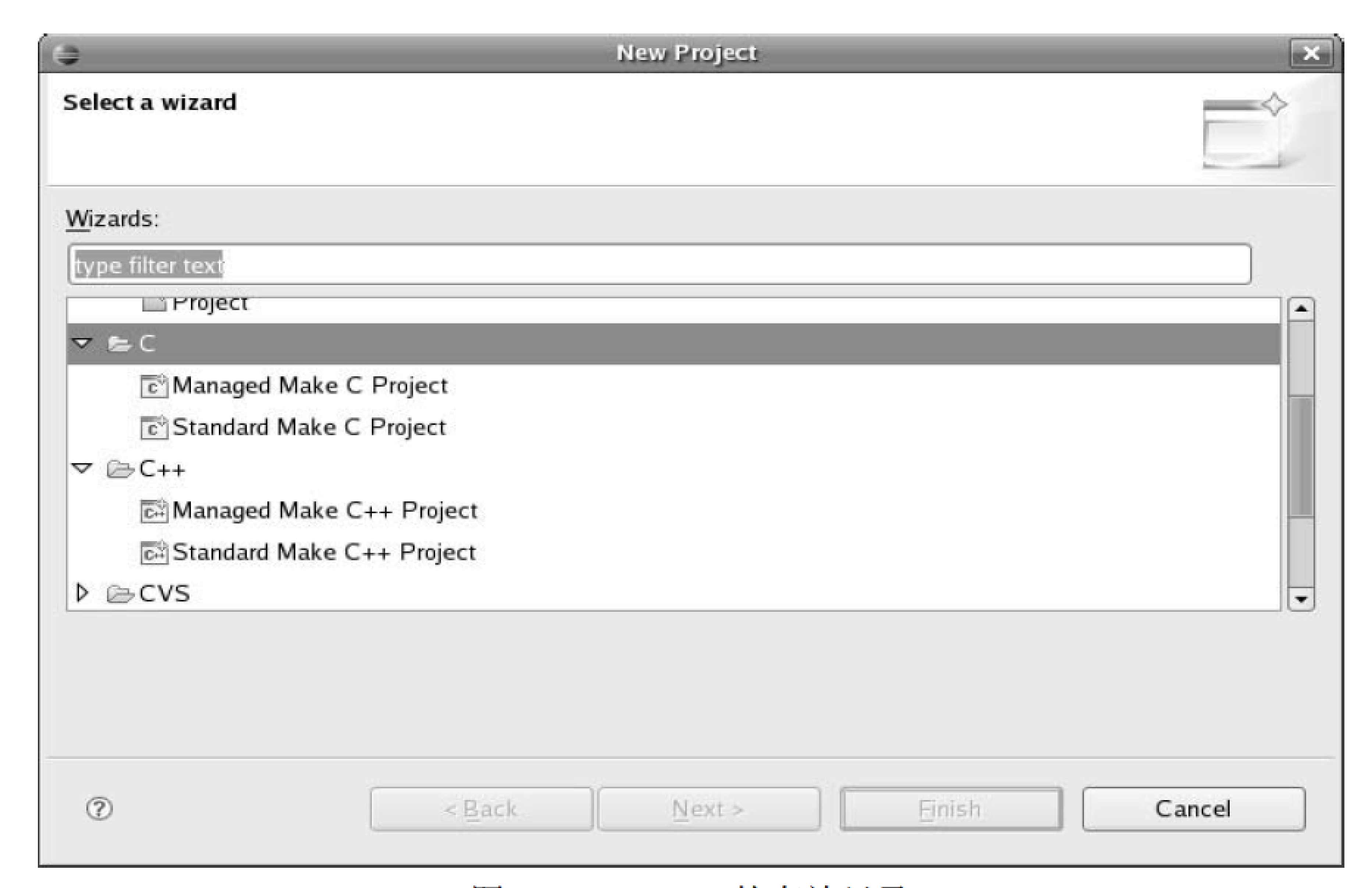


图 3-19　Eclipse 的存放目录

[root@localhost~]#cd /usr/local

(3) 复制 cdt-master-4.0.1 到当前目录：

[root@localhost~]#cp ~Desktop/CDT-master-4.0.1

(4) 解压缩：

[root@localhost~]#unzip　/CDT-master-4.0.1r

(5) 安装 CDT 插件。

将 plugins 和 festures 目录下的文件复制到 Eclipse 中相应的目录下：

[root@localhost~]#cp –r eclipse/plugins/* /usr/local/eclipse/plugins

[root@localhost~]# cp –r eclipse/features/* /usr/local/eclipse/teatures

(6) 重新启动 Eclipse，多了 C 和 C++项目支持。

4) Install PTP

(1) 下载 ptp-master-2.1.0，可将其放到桌面上。

(2) 进入存放 Eclipse 的目录：

```
[root@localhost~]#cd /usr/loca
```

(3) 复制 ptp-master-2.1.0 到当前目录：

```
[root@localhost~]#cp ~Desktop/ ptp-master-2.1.0
```

(4) 解压缩：

```
[root@localhost~]#unzip / ptp-master-2.1.0
```

(5) 安装 PTP 插件。

将 plugins 和 festures 目录下的文件复制到 Eclipse 中相应的目录下：

```
[root@localhost~]#cp –r eclipse/plugins/* /usr/local/eclipse/plugins
[root@localhost~]# cp –r eclipse/features/* /usr/local/eclipse/teatures
```

(6) 重新启动 Eclipse。

2. Windows 下 PTP 的环境建立

在 Windows 下安装 PTP 与在 Linux 下安装有稍许不同。即在安装 CDT 插件前，应该先安装 Cygwin 或 MinGW。这是因为 CDT 搭建的是一个基于开源社区 Linux 系统下的开发环境。这与 TC 编译器还是有一定不同之处的。TC 编译器是一款在 Windows 系统下工作的开发及编译工具。它们的不同之处在于对底层函数库的实现方式有所不同，最为典型的就是图形函数库，有非常本质上的区别。由于 MinGW 中没有带 GDB 调试器，所以还需要再安装一个 GDB 调试器。

在 Windows 下安装 PTP 有以下几个步骤。

1) Install Java

下载 jre-1_5_0_16-windows-i586-p 至桌面，双击进行安装。

2) Install Eclipse

下载 eclipse-SDK-3.4.1-win32 至桌面后解压缩。Eclipse 是一个绿色软件，无须安装，只需双击●图标即可运行 Eclipse。

3) Install MinGW/Cygwin

(1) Install MinGW

① 到 MinGW 官方网站上下载 MinGW 并安装，网址是 http://www.mingw.org。

② 双击 MinGW-5.1.3.exe 进行安装。

③ 单击 Next 按钮继续。

④ 选择 Download and install，单击 Next 按钮继续。

⑤ 单击 I Agree 按钮继续。

⑥ 选择 Current，单击 Next 按钮。

⑦ 根据需要选择合适的选项，这里需要选择如图 3-20 所示的选项，单击 Next 按钮。

⑧ 接下来可以根据提示依次单击 Next 按钮和 Install 按钮进行安装。

⑨ 单击 Finish 按钮完成安装。

⑩ 为了让系统能够识别 MinGW，需要设置 MinGW 的环境变量。右击“我的电脑”，选择“属性”|“高级”选项卡。

⑪ 单击“环境变量”。

⑫ 在 path 变量中加入 C:\mingw\bin(MinGW 的安装路径)。另外，由于 Eclipse 里面预设用来进行编译的文档名为 make.exe，但是 MinGW 安装后预设的 make 文件名是 mingw32-make.exe，因此需要将 mingw32-make 改名为 make。

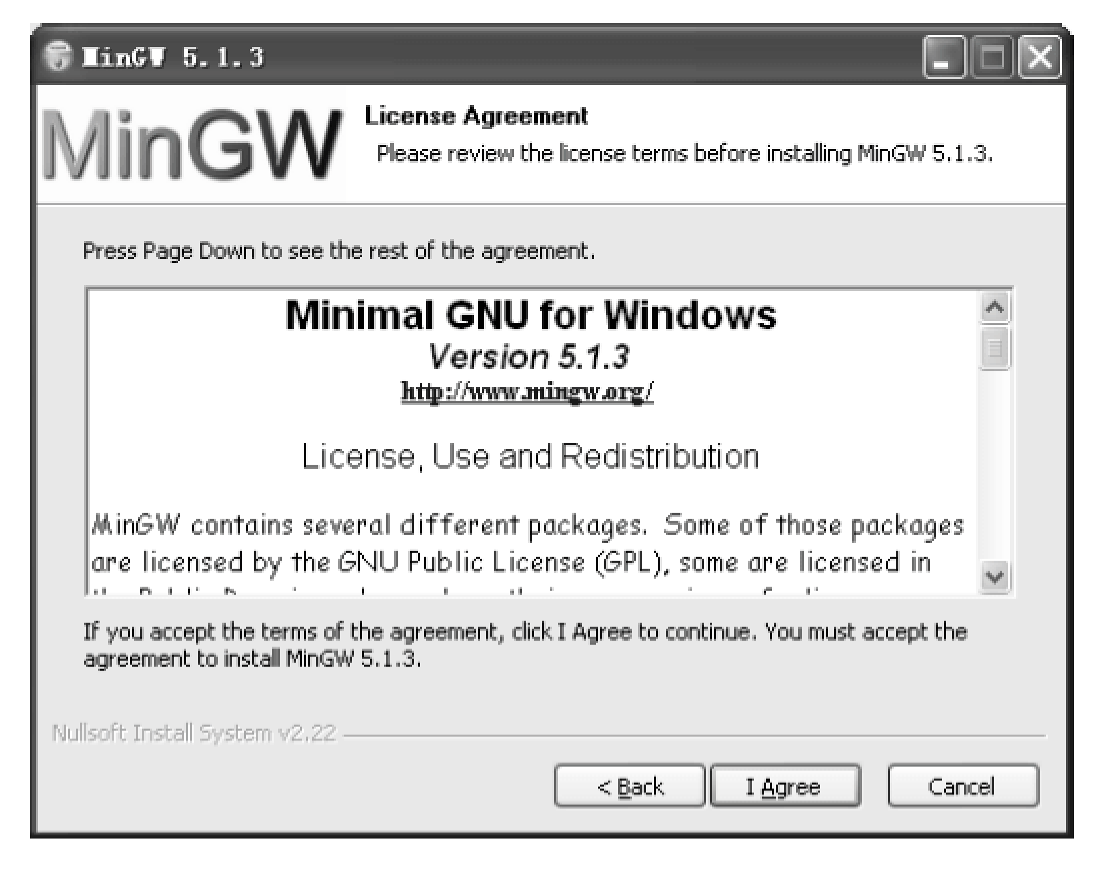
⑬ 测试 MinGW 是否安装成功：单击“开始”|“运行”，输入“cmd”，输入“gcc –v”。若出现如图 3-20 所示的信息，则说明 MinGW 安装成功。

⑭ GDB 的安装：GDB 是一个用来调试 C/C++程序的高级符号调试器。它使用户在程序运行时观察程序的内部结构和内存，由于 MinGW 中没有带 GDB 调试器，所以还需要再安装一个 GDB 调试器。在这里下载的是 gdb-5.2.1-1，然后双击即可安装。验证 GDB 是否安装成功的方法和 MinGW 一样，输入“gdb –version”。

图 3-20 是一组 MinGW 的安装图。







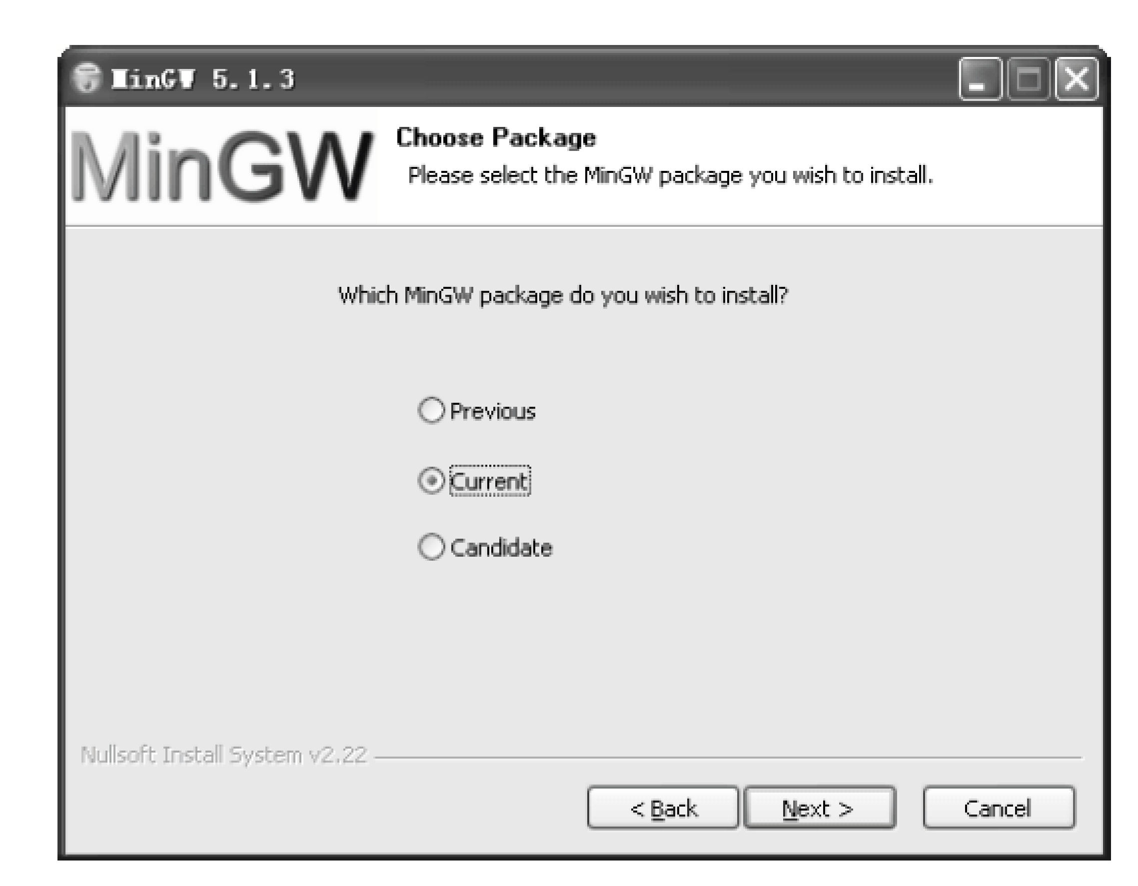
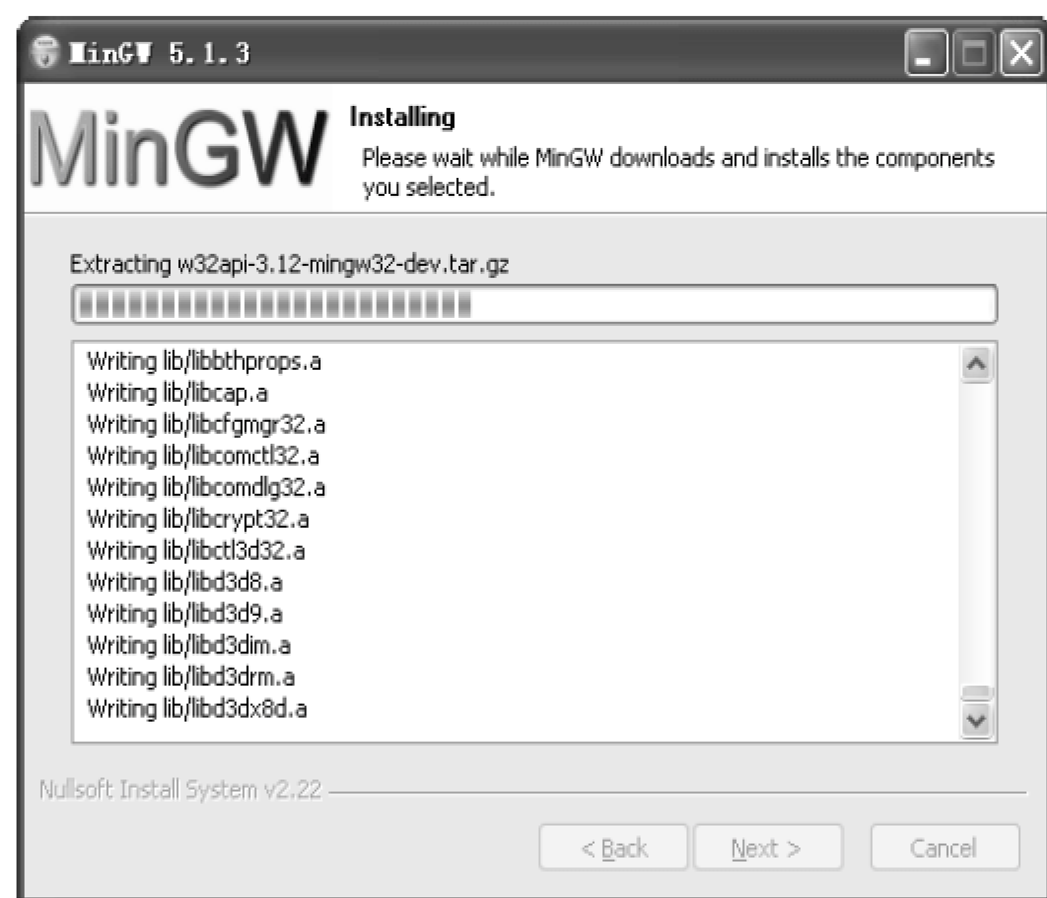
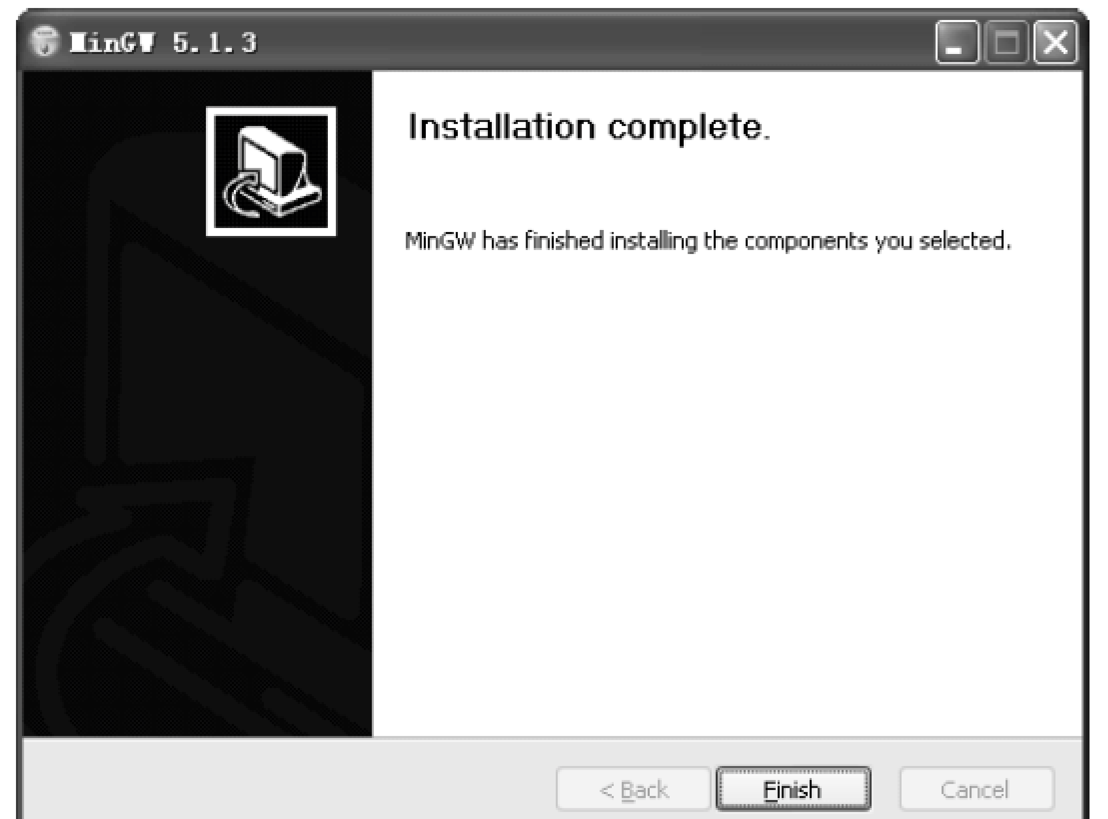
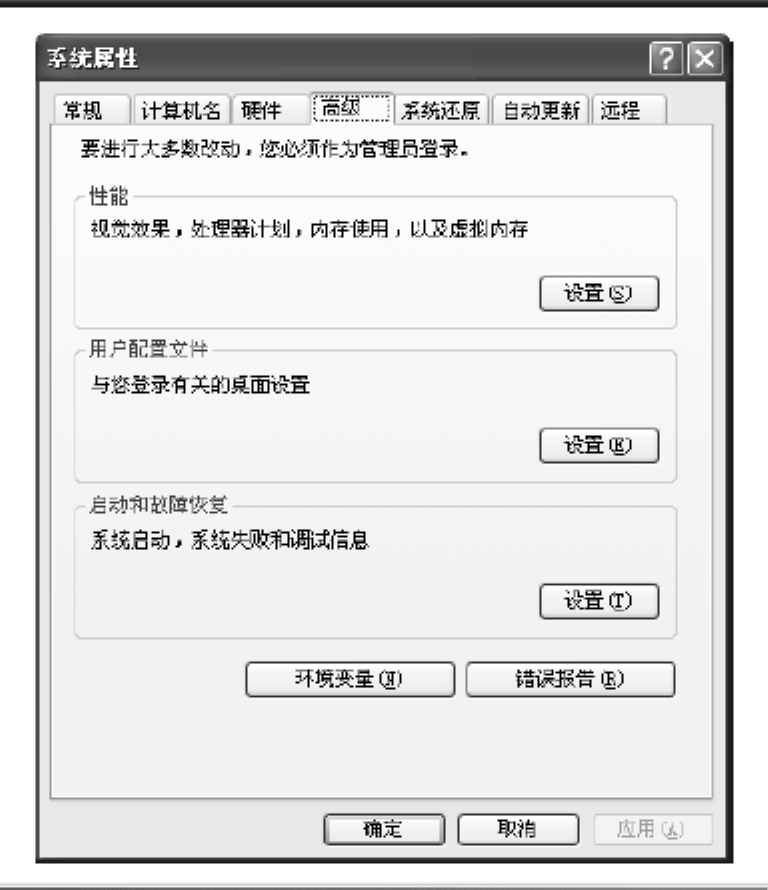


图 3-20　MinGW 的安装过程











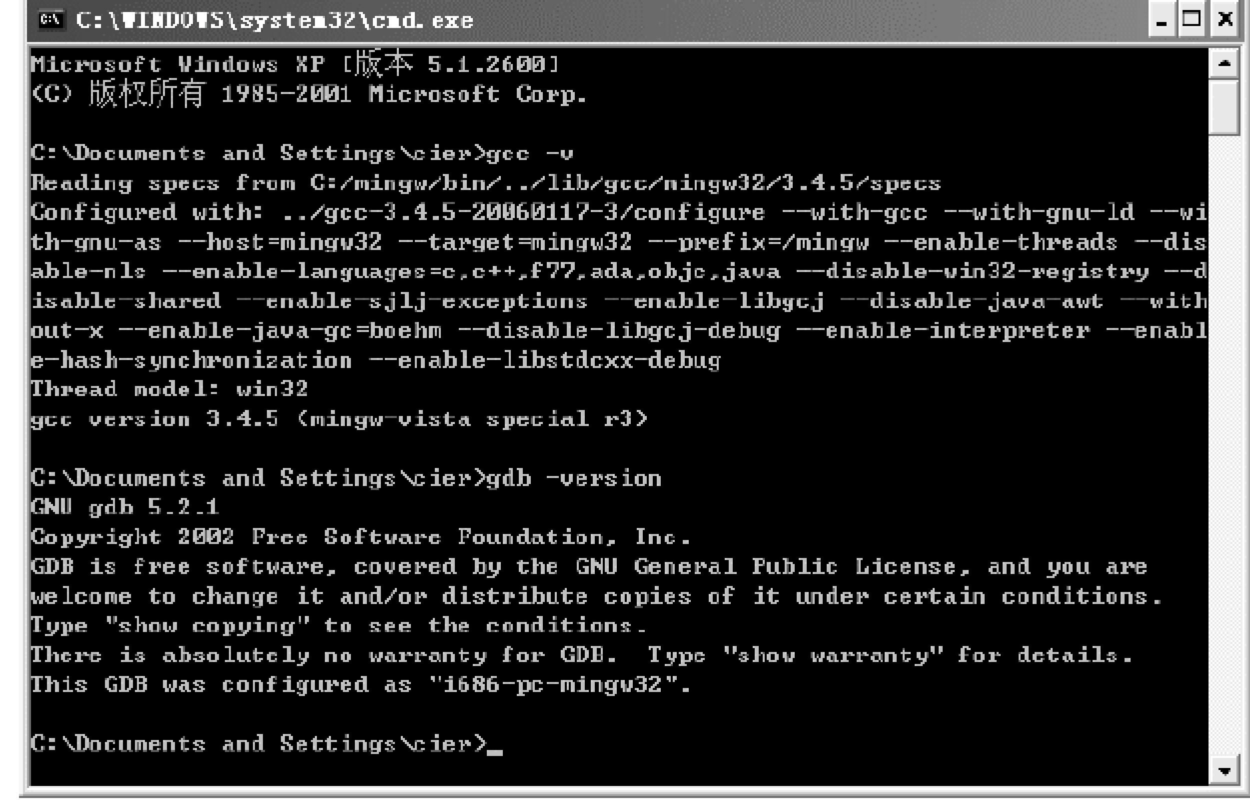


图 3-20　(续)

(2) Install Cygwin

安装过程类似于 MinGW，不同的是环境变量为 C:\Cygwin\bin。

4) Install CDT

CDT 插件在 Eclipse 下的安装有两种方法：一种是在网上下载源码包进行安装，另一种则是 Eclipse 在线升级安装，对于在线升级安装来说，Eclipse 3.4.1 已经提供了 CDT 的链接地址。

(1) 网上下载源码包安装

到 Eclipse 官方网站上下载 CDT 的压缩包，网址为 http://www.eclipse.org/cdt，如图 3-21

所示。

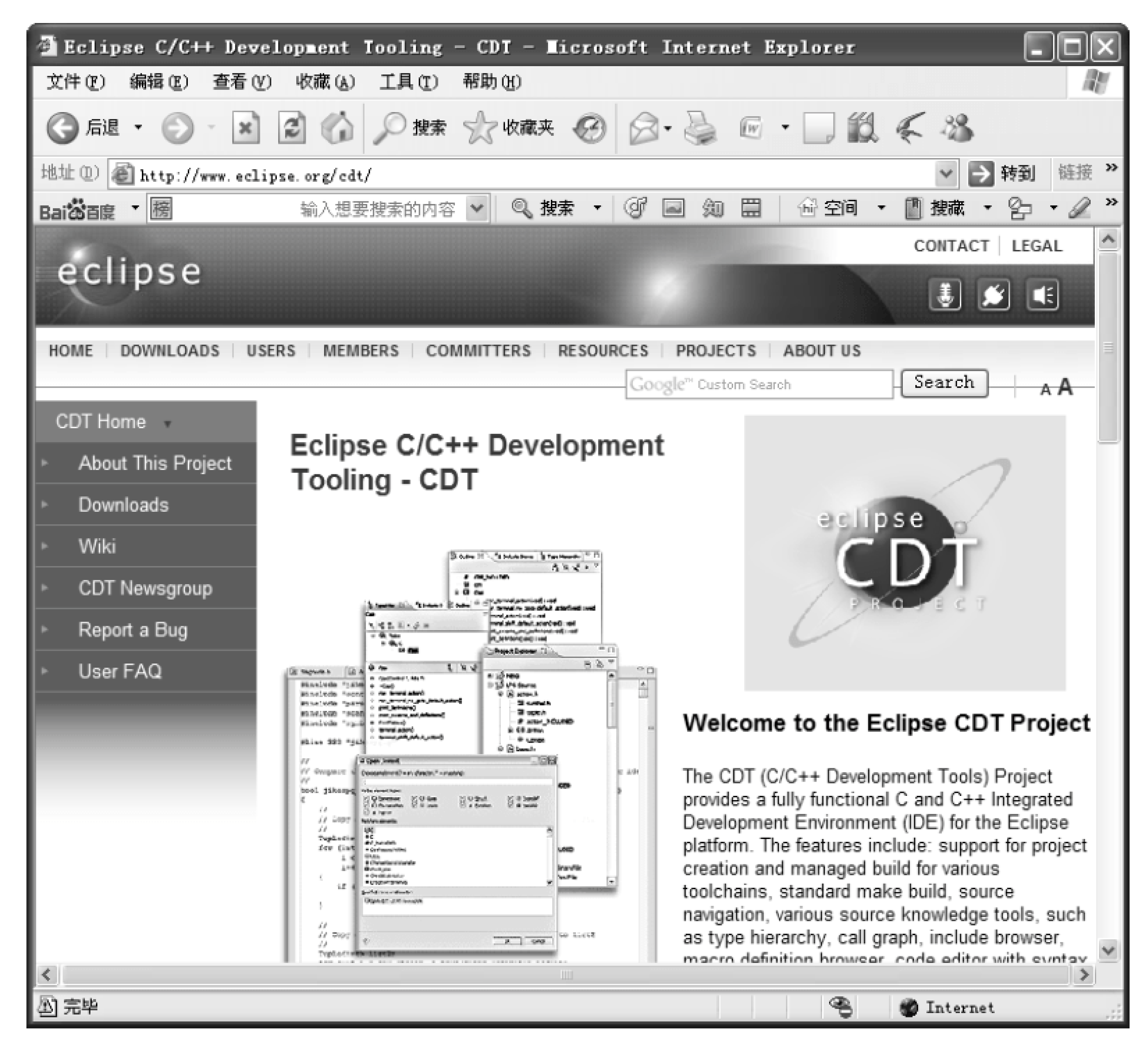


图 3-21　CDT/Eclipse 官方页面

这里选择 cdt-master-5.0.0 版本。下载并保存到 Eclipse.exe 所在的目录后解压缩，替换 features 和 plugins 两个目录下的所有文件，CDT 插件安装完毕。

(2) 在线升级安装

① 打开 Eclipse，选择 Help|Software Updates…命令，选择 Available Software。

② 选中 C and C++ Development，单击 Install 按钮，CDT 安装完成，如图 3-22 所示。

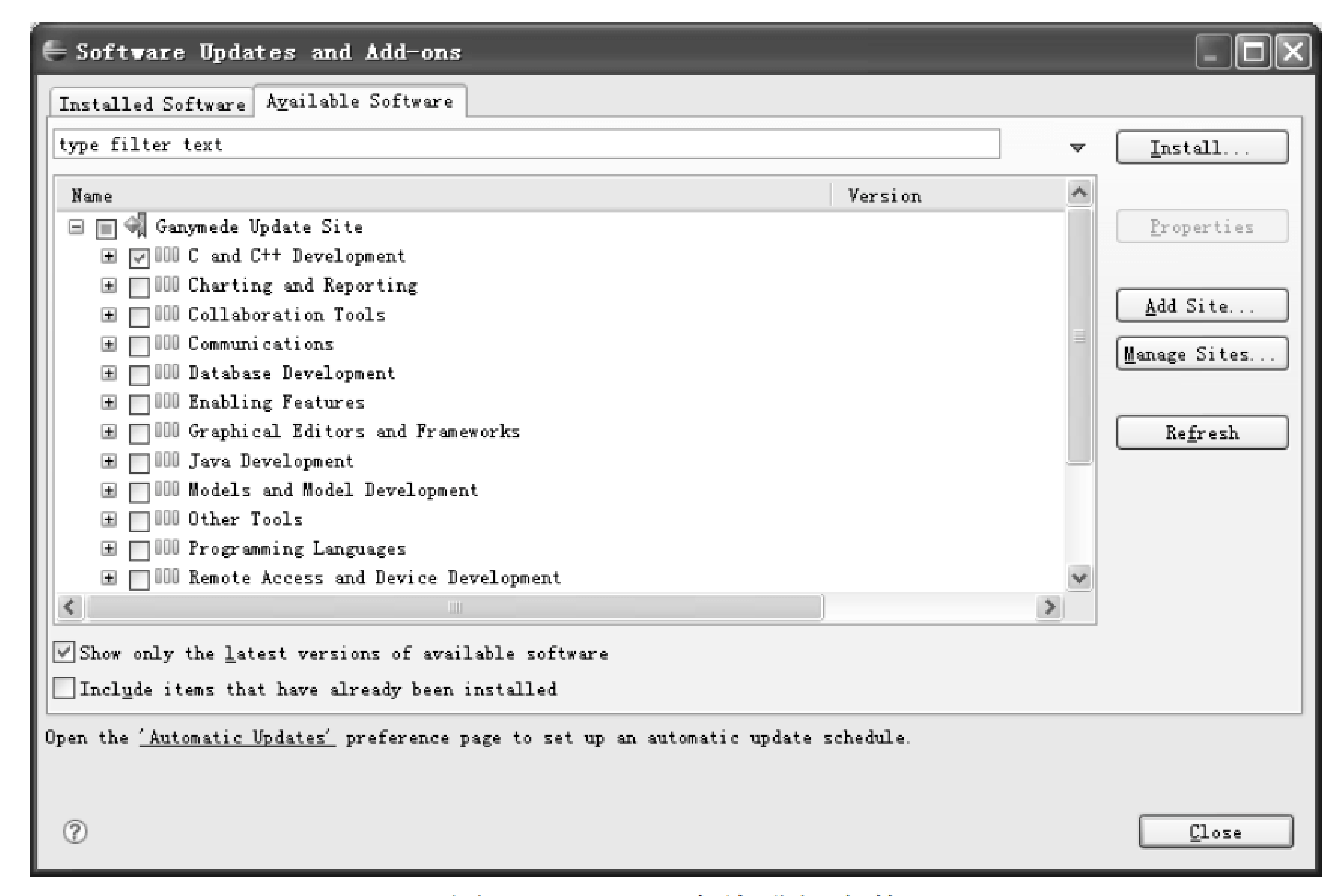


图 3-22　CDT 在线升级安装

5) Install PTP

PTP 的安装方法同 CDT 一样，一种是下载源代码包安装，另一种是在线升级安装。

(1) 网上下载源码包安装

到 Eclipse 的官方网站上下载 PTP 的压缩包，网址为 http://www.eclipse.org/ptp，如图 3-23 所示。

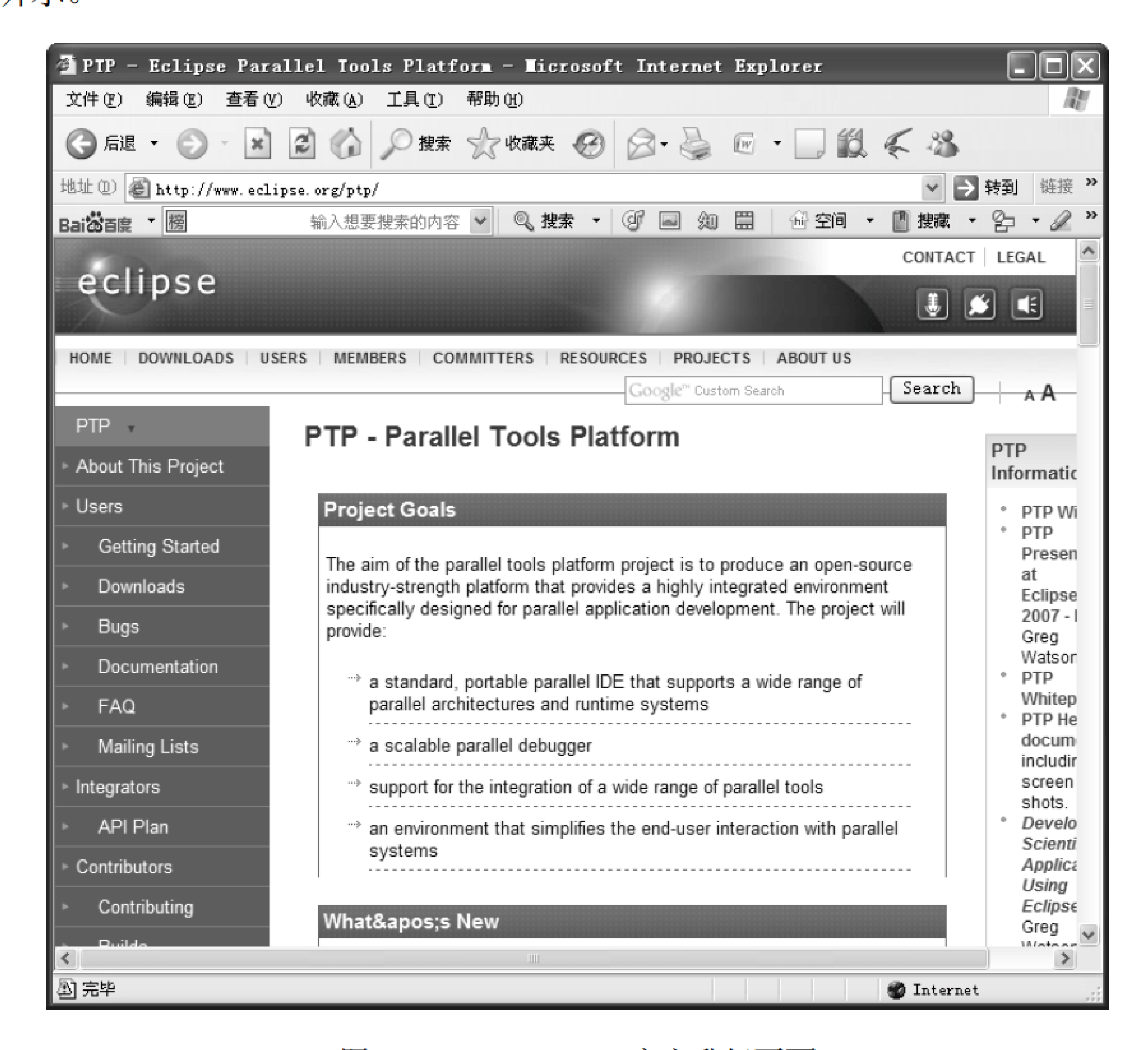


图 3-23　PTP/Eclipse 官方升级页面

这里选择 ptp-master-2.1.0-I200811031726 版本。下载并保存到 Eclipse.exe 所在的目录后解压缩，替换 features 和 plugins 两个目录下的所有文件，PTP 插件安装完毕。

(2) 在线升级安装

① 打开 Eclipse，选择 Help|Software Updates…，选择 Available Software 选项卡，如图 3-24 所示。

② 单击 Add Site…按钮，然后输入“http://download.eclipse.org/tools/ptp/releases/2.1”，单击 OK 按钮。

③ 选中“http://download.eclipse.org/tools/ptp/releases/2.1”，选择需要安装的选项。

④ 单击 Install 按钮，PTP 安装完成。

3.3.3　PTP 功能及使用流程

关于 PTP 的功能，这里仅介绍与静态分析有关的功能。PTP 可以生成控制流图、调用

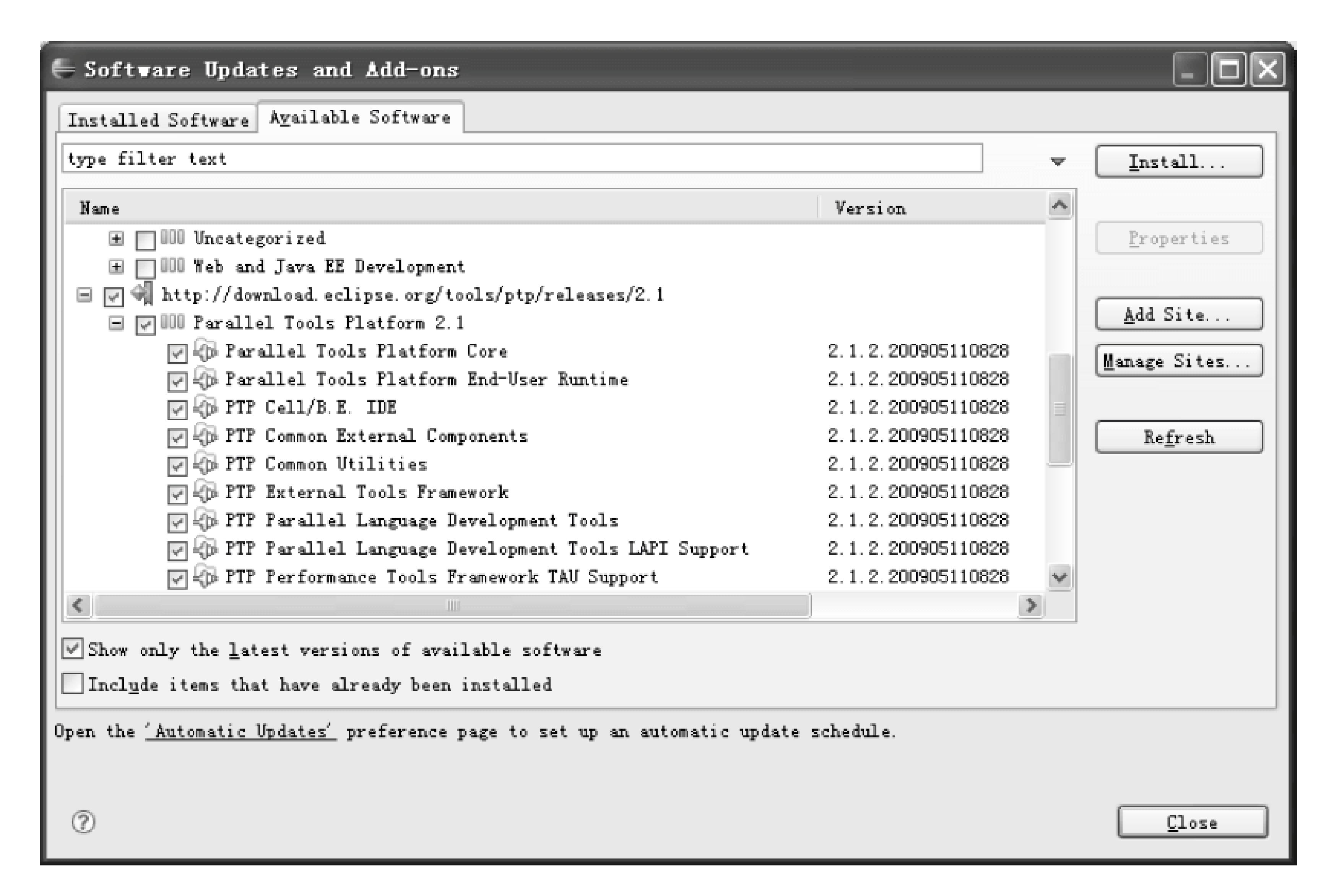


图 3-24　PTP 在线升级安装

图和依赖图，其工具栏如图 3-25 所示。



图 3-25　PTP 的工具栏

PTP 的使用流程及步骤如下。

(1) 新建一个项目或打开要分析的项目。

(2) 编译。

(3) 运行。

(4) 选择要查看的选项。

3.3.4　PTP 应用举例

示例程序：用 C 语言编写一个通讯录程序，实现添加、删除、查看、修改等基本功能。

(1) 新建一个工程，并导入光盘中的程序“通讯录程序.C”输入上面的程序，如图 3-26 所示。

在该环境下，对 C 程序进行编译，如图 3-26 所示。图的右边是程序的大纲，包括程序包含的头文件、定义的全局变量、定义的结构体、结构体变量、定义的函数以及实现的函数。通过这个大纲，可以看到程序定义了哪些全局变量、哪些结构体、有哪些函数虽然定义了但却没有用以及有哪些函数是没有定义的，这可以使我们对程序有个大概的了解。

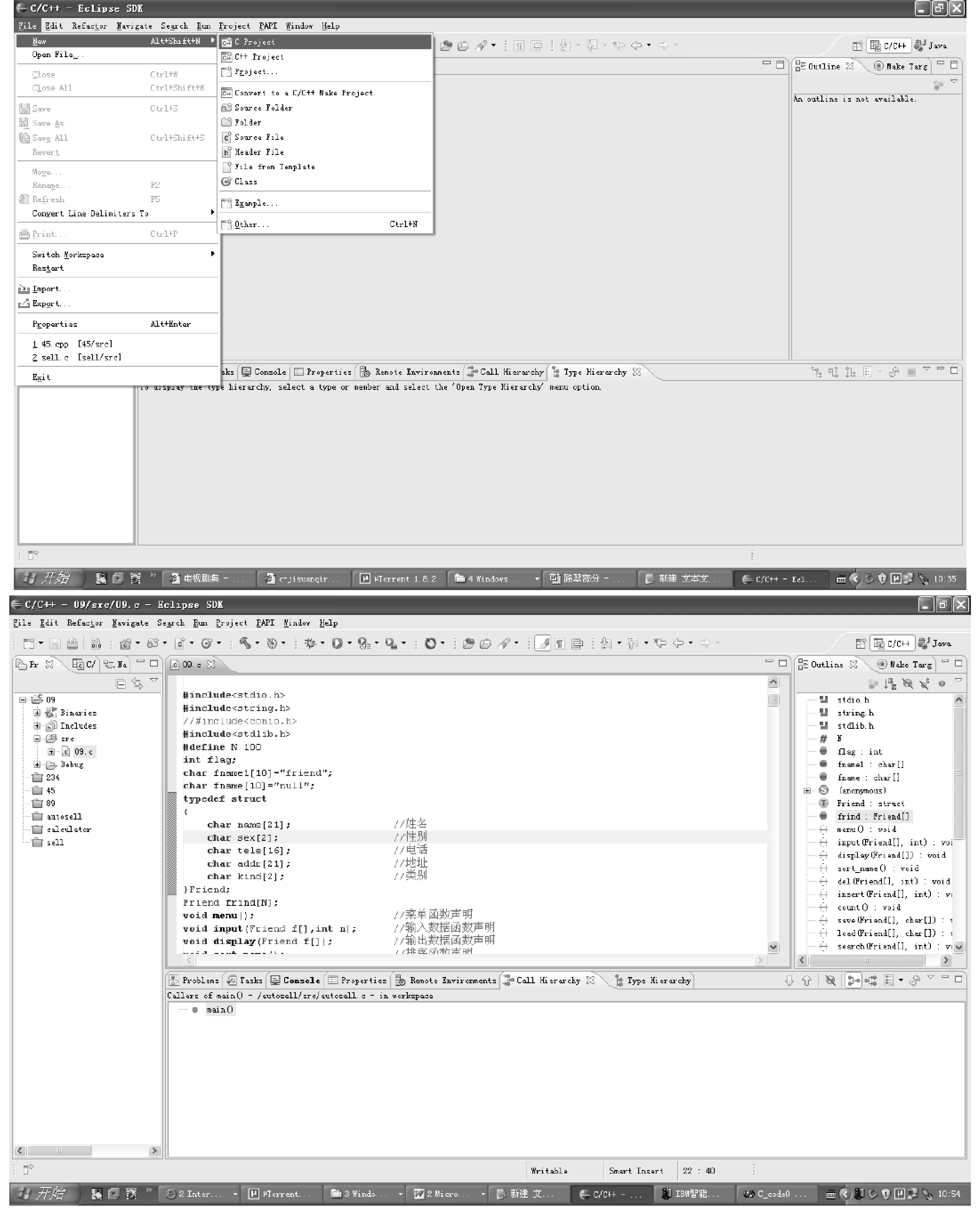


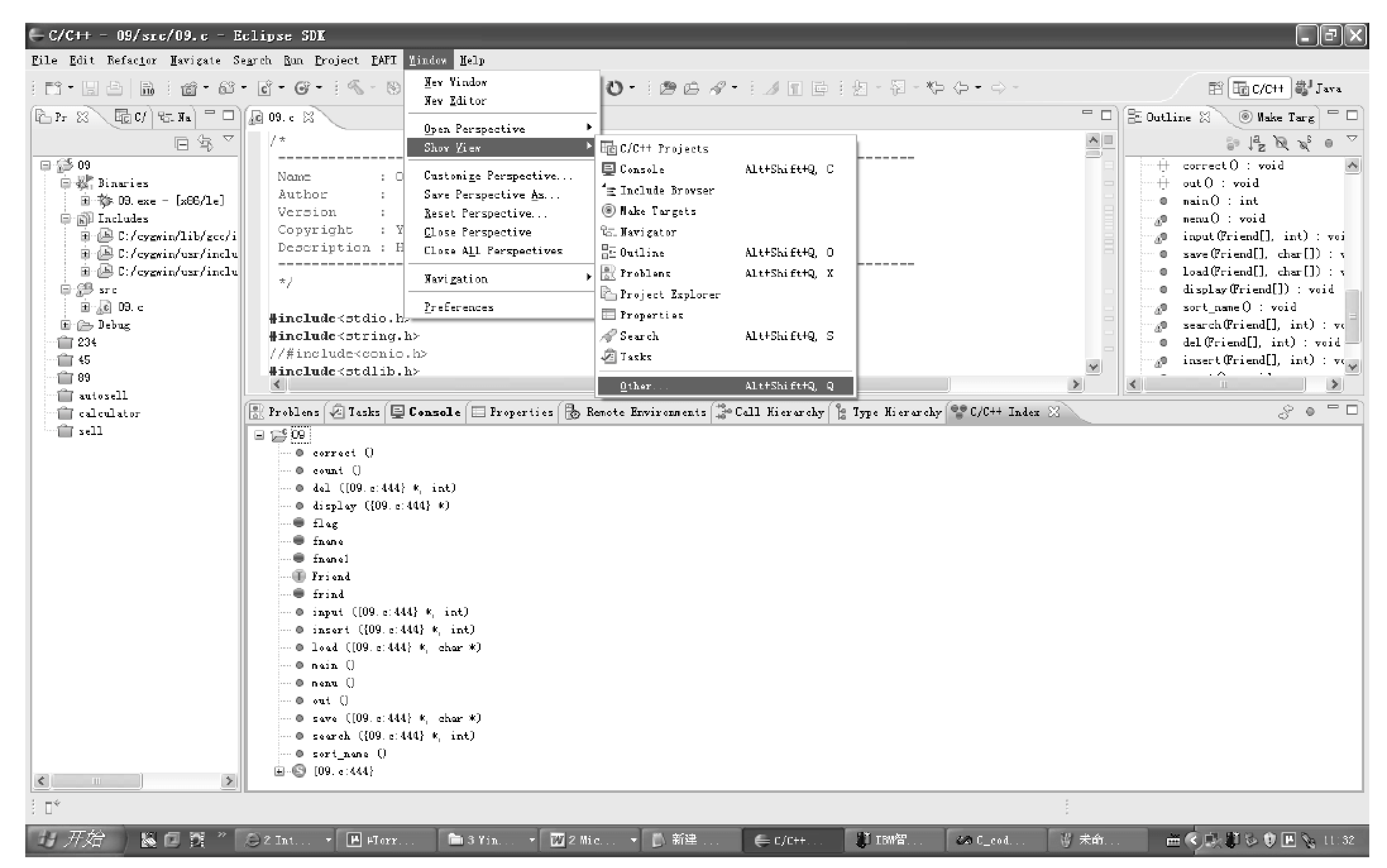
图 3-26　新建工程、编辑程序

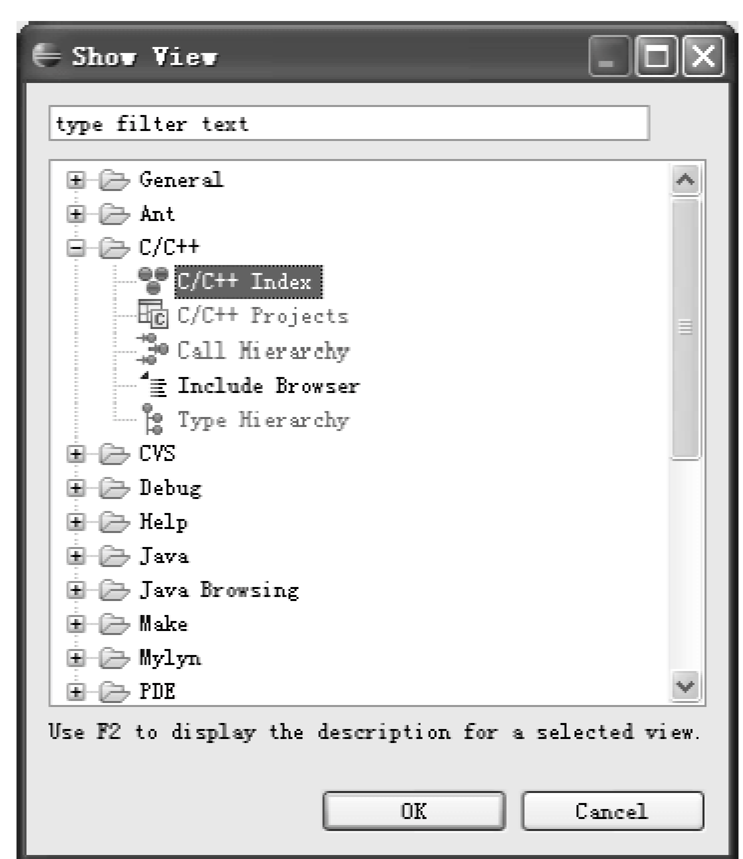
(2) 当程序太大时，右边的大纲就会显得比较凌乱。这时可以借助 PTP 提供的索引功能来查看程序中变量及函数的定义。

打开程序索引的方法是：①选择 Window|Show View|Others 命令；②弹出一个窗口，选择 C/C++ Index；③单击 OK 按钮，然后程序主界面的下方会出现名为“C/C++ Index”的选项卡及程序的索引，以方便用户从整体上了解程序，整个过程如图 3-27 所示。

通过索引窗口可以清楚地看到每一个函数、变量及类型定义等。其中，绿色的原点

表示函数；蓝色的原点 表示变量； 表示定义的结构体，单击图标前面的加号即可看到结构体所包含的成员； 表示定义的新类型。





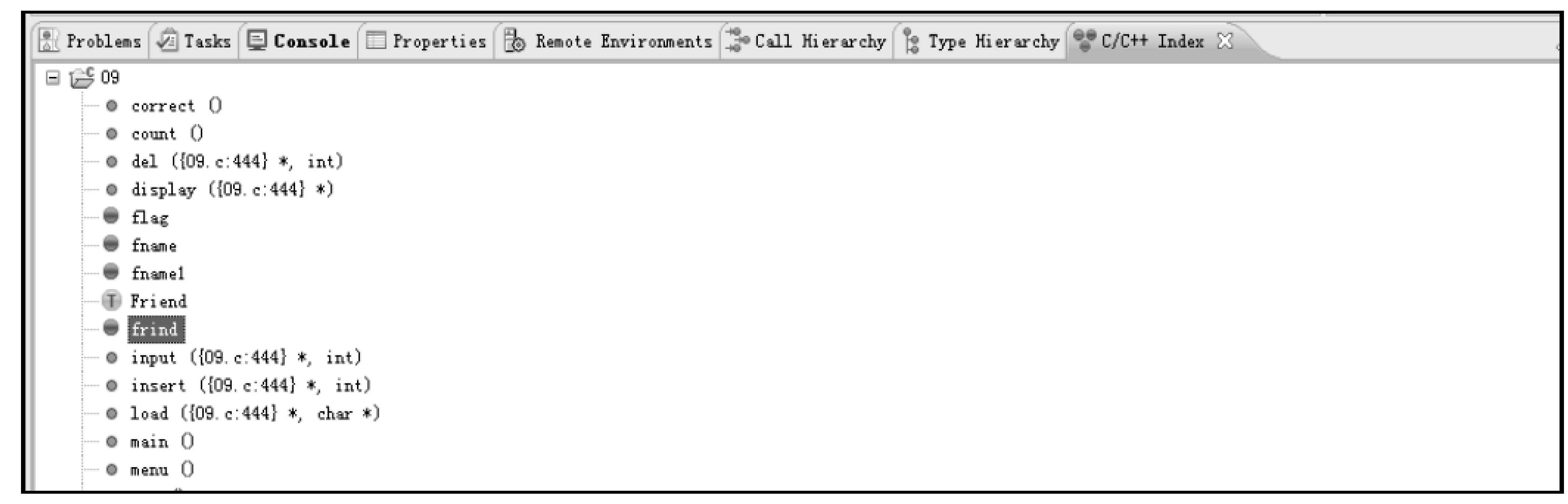


图 3-27　打开程序索引

(3) 在编译执行后，可以查看函数调用图。例如，想查看函数 Load 被哪些函数调用了，可以选中函数 Load 的名字，然后右击，选择 Open Call Hierarchy 命令或者单击 按钮，

便可以看到哪些函数调用了 Load 函数。如图 3-28 所示，可以看到在函数 correct()、count()、del()、display()、insert()、search()和 sort_name()中均调用了函数 Load。

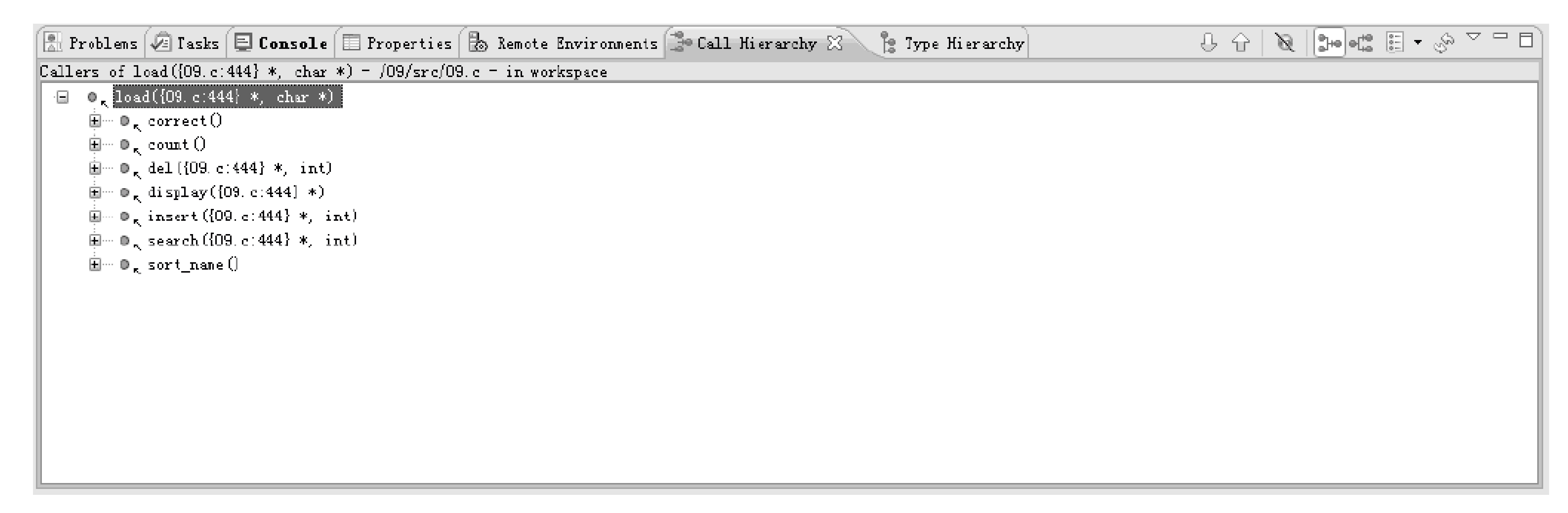


图 3-28　查看函数调用图

如果想同时查看函数 Load 内部调用了哪些函数，单击按钮，就可看到函数 Load 调用了 fopen()、printf()、exit()等函数，如图 3-29 所示(注：前面图标为和的为函数)。

图 3-29　查看函数内部调用图

(4) 另外，还可以查看变量的类型、继承关系、组成元素等。

例如，查看示例程序中 Friend 的类型、继承关系及组成元素，可以选中 Friend，然后右击，选择 Open Type Hierarchy 命令即可，如图 3-30 所示。

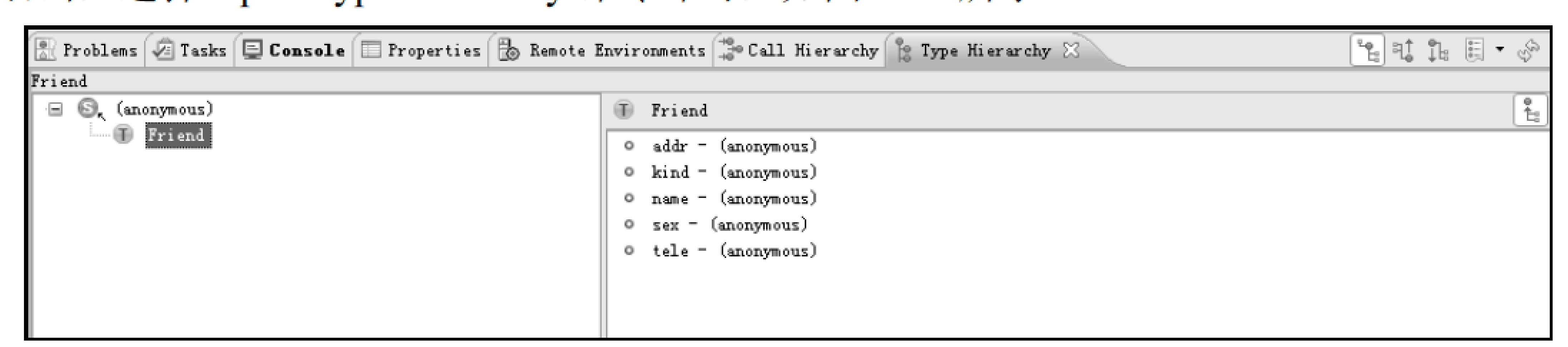


图 3-30　查看变量的相关信息

(5) 最后，可以查看程序的 AST。

首先要显示 DOM AST 窗口。打开 DOM AST 窗口的方法与打开以上窗口的方法一致，如图 3-31 所示。

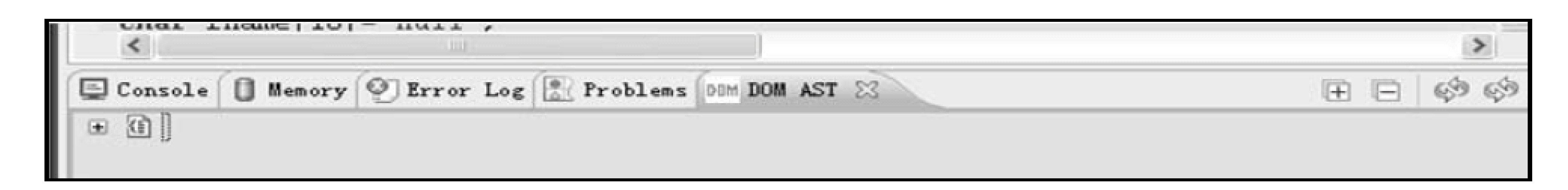


图 3-31　打开 DOM AST 窗口

然后在任意函数或变量上右击鼠标，选择 Show IASTNode in Dom View 选项，即可出现 AST，如图 3-32 所示。每一行代表一个节点，“+”表示此节点下还有子节点。这些节点组成了 AST(抽象语法树)。

图 3-32　AST 语法树信息

同时，当选中其中一个节点时，程序中相应的文件、函数、变量、宏定义等就会呈选中状态，如图 3-33 所示。例如，在 DOM AST 窗口中选中 stdio.h 节点，则源程序中“#include <stdio.h>”将呈选中状态。

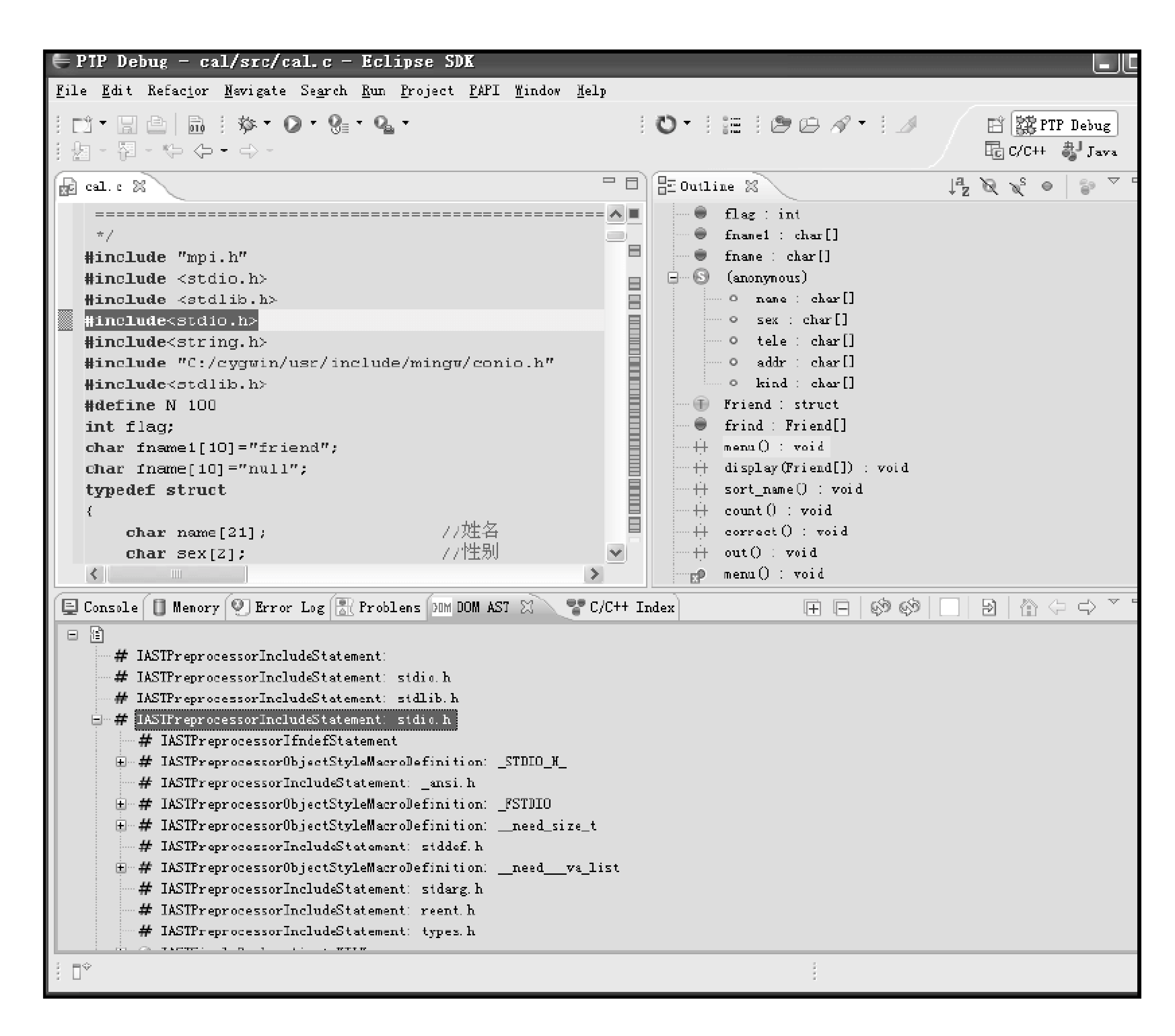


图 3-33　选中 stdio.h 节点

展开 stdio.h 对应节点，即可查看 stdio.h 文件中的宏定义、变量等。例如，随意单击其中一个节点，将出现如图 3-34 所示情况，这里选中了 stdio.h 中自定义的新类型 FILE。

(6) 最后可以应用 PLDT（并行语言开发工具，它是 PTP 的分析部分）查看程序语句

级的控制流图 CFG，如图 3-35 所示创建控制流，图 3-36 为控制流图示例，图 3-37 为生成控制流图正文表示。

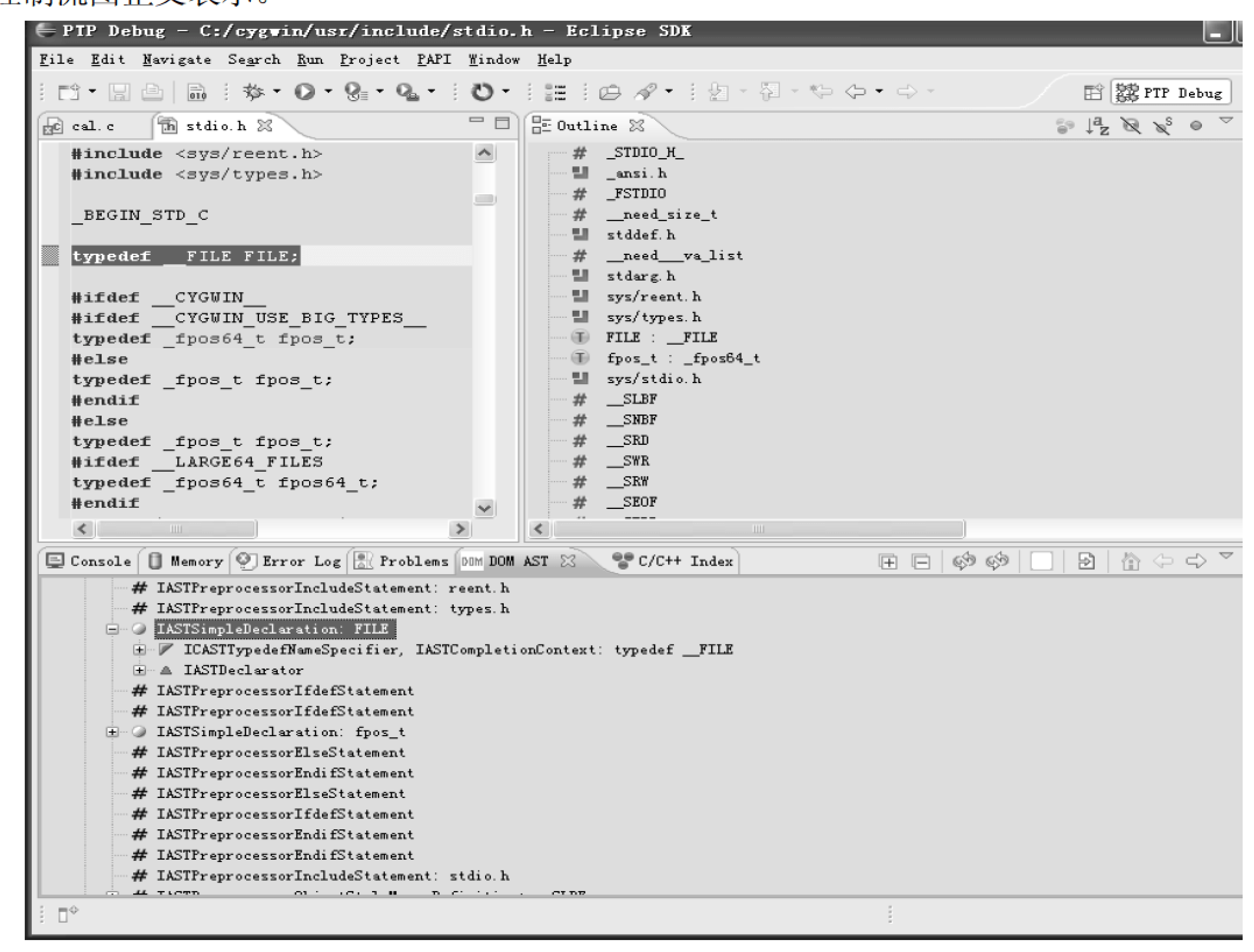
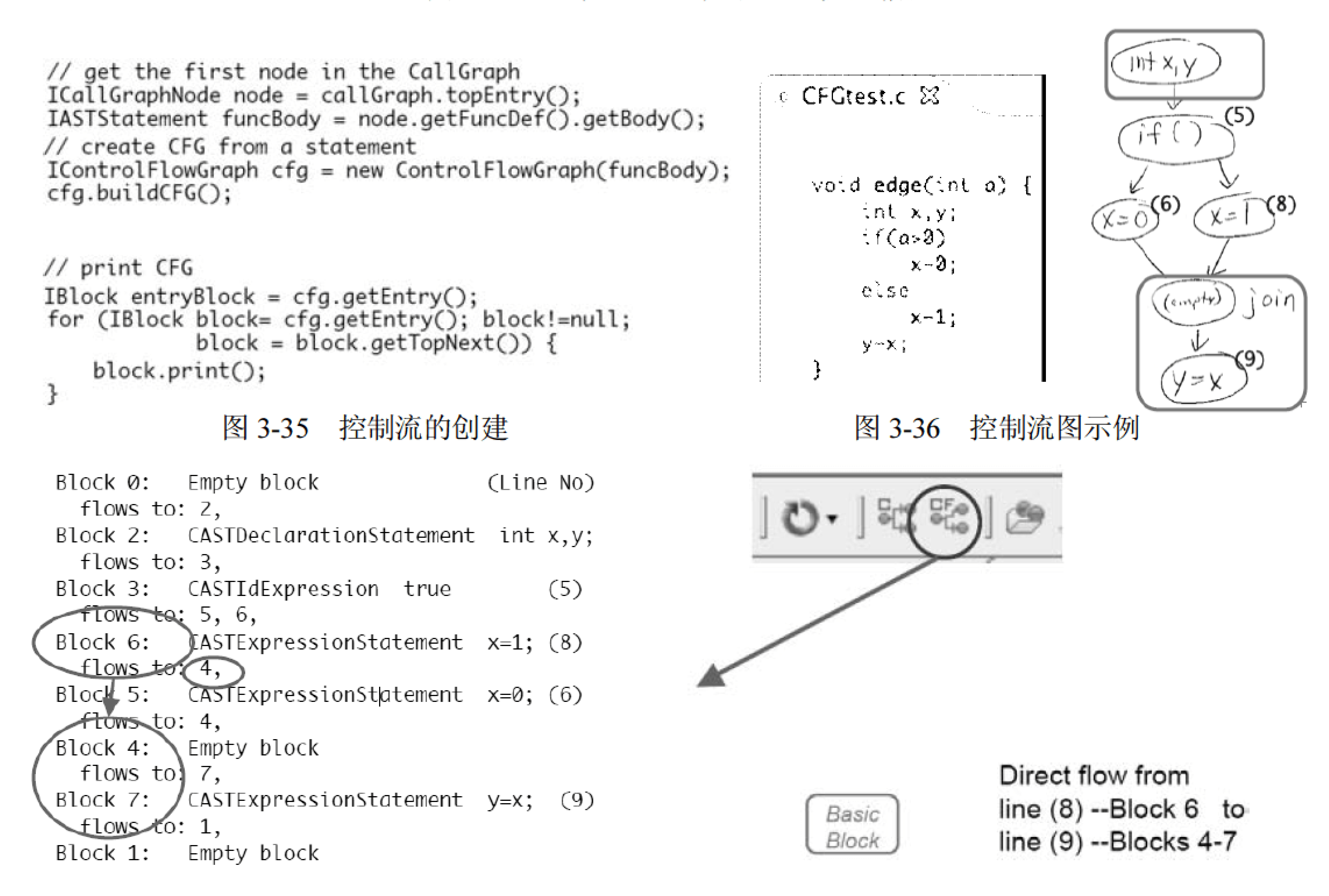


图 3-34　显示 stdio.h 中的 FILE 类型信息



```
// get the first node in the CallGraph
ICallGraphNode node = callGraph.topEntry();
IASTStatement funcBody = node.getFuncDef().getBody();
// create CFG from a statement
IControlFlowGraph cfg = new ControlFlowGraph(funcBody);
cfg.buildCFG();

// print CFG
IBlock entryBlock = cfg.getEntry();
for (IBlock block= cfg.getEntry(); block!=null;
            block = block.getTopNext()) {
    block.print();
}
```

图 3-35　控制流的创建

图 3-36　控制流图示例

```
Block 0:    Empty block                (Line No)
  flows to: 2,
Block 2:    CASTDeclarationStatement  int x,y;
  flows to: 3,
Block 3:    CASTIdExpression  true          (5)
  flows to: 5, 6,
Block 6:    CASTExpressionStatement   x=1;  (8)
  flows to: 4,
Block 5:    CASTExpressionStatement   x=0;  (6)
  flows to: 4,
Block 4:    Empty block
  flows to: 7,
Block 7:    CASTExpressionStatement   y=x;   (9)
  flows to: 1,
Block 1:    Empty block
```


图 3-37　控制流的正文表示生成

实 验 习 题

1. 分别使用 Oink 和 PTP/CDT/Eclipse 对一个具体的 C/C++程序进行理解分析，并完整地给出相关的图形表示。

2. 列举其他支持程序理解的开源工具或共享工具，并给出它们详细应用的各种结果。

第4章　代码静态分析工具

代码静态分析主要是进行代码的检查、代码结构的分析、代码问题的查找和代码质量的度量。它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具进行。代码静态分析主要是分析和检查代码与设计的一致性，代码对标准的遵循，以及代码的可读性、代码逻辑表达的正确性、代码结构的合理性等方面；可以发现违背程序编写的标准问题，程序中不安全、不明确和模糊的部分，找出程序中不可移植部分、违背程序编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

代码静态分析工具能够帮助人们保证代码的质量，发现并警告代码中潜在的错误。代码静态分析工具和编译器的某些功能其实是很相似的，它们都是利用编译器的前端功能进行词法分析、语法分析、语意分析，并收集各种用于代码分析的结果。代码静态分析工具和编译器的不同之处在于，它们可以自定义各种各样的复杂的规则来对代码进行分析，并利用先进的图形表示手段给出各种分析的图形结果。

代码静态分析工具的实现会因为实现方法、算法、分析的层次不同，功能上差异很大。总地来说，商用的代码静态分析工具功能齐全、完整，分析结果的可视化效果好，在经济条件许可的情况下是最佳选择；开源的代码静态分析工具尽管功能受限，但在某些方面它们还是有各自优势的。另外，目前有很多 IDE 已经将很多可以用于代码静态分析的功能紧密地集成在了一起，甚至提供了插件的接口来扩展其代码静态分析的能力。

4.1　代码静态分析工具及编程规范检查

错误发现得越晚，修正的成本就越高，测试阶段修正错误的成本约是编码阶段的 4 倍。为了减少成本，错误被发现得越早越好。在编程阶段，通过静态分析找出代码中大部分的错误，是很多人的梦想。这个梦想在 21 世纪初变成了现实。目前 IT 业界已经在大量使用代码静态分析工具，以便在编码阶段就能够找出可能的编码缺陷。以 KlocWork 公司的 K7、Coverity 公司的 Prevent、Parasoft 公司的 Insure++、Fortify Software 公司的 SCA 和 Gimpel Software 公司的 PC-Lint 等为代表的商用静态分析软件，以及以 Prefast、Cppcheck、Findbugs、PMD、Checkstyle、Splint 等为代表的开源静态分析工具，实现了只要静态分析代码，就可以发现代码的错误，例如数组越界、除数为 0、缓冲区溢出等。尽管它们或多或少地存在一些问题，还不是特别完美，但对人们的帮助还是相当大的。下面对一些开源的代码静态

分析工具做一般性的介绍。

4.1.1　静态代码分析工具介绍

1. C/C++代码静态分析工具 Prefast

Prefast 是一种代码静态分析工具，它能够帮助用户找到编译器不能找到的错误或缺陷。Prefast 还首次被微软集成到 Visual Studio 2005 Team Suite 中去，使用起来非常方便。

Prefast 是由微软研究院提出的代码静态分析工具。其主要目的是通过分析代码的数据和控制信息来检测程序中的缺陷。需要强调的是，Prefast 检测的缺陷不仅是安全缺陷，但是安全缺陷类型是其检测的最为重要的部分。Prefast 推出后在微软内部得到了广泛使用，并经历了若干个版本的升级。现在，微软将这个内部工具商业化，提供给外部的开发人员使用。

目前有以下两个办法可以获得 Prefast 工具。

(1) Prefast 包含在 Visual Studio 2005 /2008 的团队版本(Team Edition)中。

(2) Prefast 包含在 Windows 驱动程序开发包(Microsoft Windows Driver Kits)的开发环境中。

需要指出的是，Visual Studio 团队版的价格要高于 Visual Studio 个人版，而 Windows 驱动程序开发包是免费下载的。那么，它们提供的 Prefast 版本有什么区别？在 Visual Studio 团队版中，Prefast 是直接和代码的开发过程集成的，使用非常方便，并且可以直接根据 Prefast 的输出结果创建相应的开发任务。而在 Windows 驱动程序开发包中，Prefast 作为一个单独的工具提供，而不像 Visual Studio 团队版中一样与开发环境集成。要下载 Windows 驱动程序开发包，可以进入网站 http://connect.microsoft.com/，注册 Microsoft Connect 后选择 Windows Logo Kit (WLK)、Windows Driver Kit (WDK)和 Windows Driver Framework (WDF)即可。具体步骤这里就不详细叙述了。

安装好 WDK 后，Prefast 就可以直接在其开发环境下使用了。

在 VS 2005 Team Suite 中，Prefast 使用起来非常简单。修改工程属性，设置 Enable Code Analysis For C/C++为 Yes 即可。

Prefast 的主要检查内容如下：初始化、空指针、内存泄漏、=与= =的误用、安全问题、字符串错误、死循环等。Prefast 会将自动检查出的错误在编辑器中显示成浅灰色。

2. FindBugs、PMD 和 CheckStyle

主要用于 Java 代码的检查，表 4-1 中给出了各个工具的检查侧重点。

3. 其他代码静态分析工具

1) Valgrind

Valgrind 软件包是开源的，可以从 www.valgrind.org 上下载到，并有很详细的用户文档。Valgrind 可帮助程序员寻找程序里的 Bug 和改进程序的性能。程序通过 Valgrind 运行时，Valgrind 将收集各种有用的信息，通过这些信息可以找到程序中潜在的 Bug 和性能瓶颈。

表 4-1　FindBugs、PMD 和 CheckStyle 的检查侧重点

工　具	目　的	主要检查内容
FindBugs	基于 Bug Patterns 概念，查找 Java bytecode 中的潜在 Bug。在目前版本中，它不检查 Java 源文件	主要检查 bytecode 中的 Bug Patterns，也允许用户自定义特定的 Bug Patterns
PMD	检查 Java 源文件中的潜在问题	主要包括： 空 try/catch/finally/switch 语句块 未使用的局部变量、参数和 private 方法 空 if/while 语句 过于复杂的表达式，如不必要的 if 语句等 复杂类
CheckStyle	检查 Java 源文件是否与代码规范相符	主要包括： Javadoc 注释 命名规范 Headers Imports Size 冲突和度量，如过长的方法 Whitespace Modifiers Blocks Coding Problems Class Design 重复代码 Miscellaneous Checks Optional Checks

Valgrind 提供多个很重要的工具：Memcheck、Cachegrind、Massif 和 Callgrind。Valgrind 可用于在 Linux 系统下开发应用程序时调试内存问题，尤其擅长发现内存管理方面的问题，它可以检查程序运行时的内存泄漏问题。其中最有用的就是内存检测工具 Memcheck，它可以检测出许多 C/C++程序中常见的内存错误，如：

(1) 内存越界访问。

(2) 使用未初始化的变量。

(3) 将无意义的参数传递给系统调用。

(4) 错误的内存释放，比如两次释放同一内存块。

(5) 内存泄漏。

2) Cppcheck

Cppcheck 是静态的 C/C++ 代码分析工具，用于检查内存泄漏、错误的内存分配和释放、缓冲区溢出，以及其他更多的问题。

3) Beam

Beam 可以检测以下 4 类问题：未初始化的变量；废弃的空指针；内存泄漏；冗余计

算。而且 Beam 支持的平台也比较多，Beam 支持以下平台。

(1) Linux x86 (glibc 2.2.4)

(2) Linux s390/s390x (glibc 2.3.3 or higher)

(3) Linux (PowerPC，USS) (glibc 2.3.2 or higher)

(4) AIX (4.3.2+)

(5) Window 2000 or higher

4) Lint

如果想用一个有效的工具查看 C/C++源代码中的错误、遗漏、不确定的构建过程，以及移植问题等，那么应该考虑 Lint。可以把 Lint 当成一个编译器，除了不产生代码之外，对于错误和警告的报告来说已经足够了。

通常，一个 C/C++的编译器假设程序是正确的，而 Lint 恰恰相反，因此它优于编译器执行的一般性检查。Lint 还可以贯穿多个文件来执行它的错误检查和代码分析，这是编译器做不到的。

下面是 Lint 能够检查的部分错误列表。幸运的话，用户的编译器也可以检查出其中的一些，但不会是全部。

(1) 可能的空指针。

(2) 在释放内存之后使用了指向该内存的指针。

(3) 赋值次序问题。

(4) 拼写错误。

(5) 被零除。

(6) 失败的 case 语句(遗漏了 break 语句)。

(7) 不可移植的代码。

(8) 宏定义参数没用使用圆括号。

(9) 符号的丢失。

(10) 异常的表达式。

(11) 变量没有初始化。

(12) 可疑的判断语句(例如，if (x = 0))。

(13) printf/scanf 的格式检查。

现有的 Lint 程序主要有两个版本：① PC-Lint 是一个由 Gimpel Software 提供的支持 C/C++的商用程序；② Splint(原来的 LCLint)是一个 GNU 免费授权的 Lint 程序，但是只支持 C 而不支持 C++。

Lint 的运行：运行 Lint 与运行一个正常的编译器一样，只要把编译命令直接加入 makefile 中就可以了。

下面将专门介绍 PC-Lint 和 Splint 工具。PC-Lint 是商用软件，这里介绍它主要是因为它的普及和物美价廉。

4.1.2　编程规范检查工具 CheckStyle

CheckStyle 是非常优秀的代码规范检查软件，可以大幅地提高代码质量，当项目的开发人员比较多时，用它来统一代码风格是很有必要的。

它可以根据设置好的编码规则来检查代码。比如符合规范的变量命名，良好的程序风格等。如果项目经理开会时说，“我希望我们写出来的代码就像一个人写的！”时，用 CheckStyle 绝对是正确选择。

需要强调的是，CheckStyle 只能做检查，而不能修改代码。想修改代码格式，请使用 Jalopy。它和 CheckStyle 配合使用非常合适。

CheckStyle 的配置性极强，可以只检查一种规则，也可以检查三四十种规则。可以使用 CheckStyle 自带的规则，也可以自己增加检查规则。它支持几乎所有主流 IDE，包括 Eclipse、IntelliJ、NetBeans、JBuilder 等。Eclipse 的 CheckStyle 插件下载地址为 http://sourceforge.jp/projects/sfnet_eclipse-cs/releases/。

1. 插件的安装

首先这个插件不是 Eclipse 自带的，而是需要去网上下载的，得到插件之后在 Eclipse 的 links 文件夹中新建一个 checkstyle_4.2.link 的文件，以记事本打开，在里面设置一个 path 来记录插件的安装路径：path=D://tool//checkstyle_4.2。注意路径用“//”隔开，否则不会显示出插件的内容。在 Eclipse 的 Window|Preferences 中可以看到 Checkclipse 的选项，如图 4-1 所示。

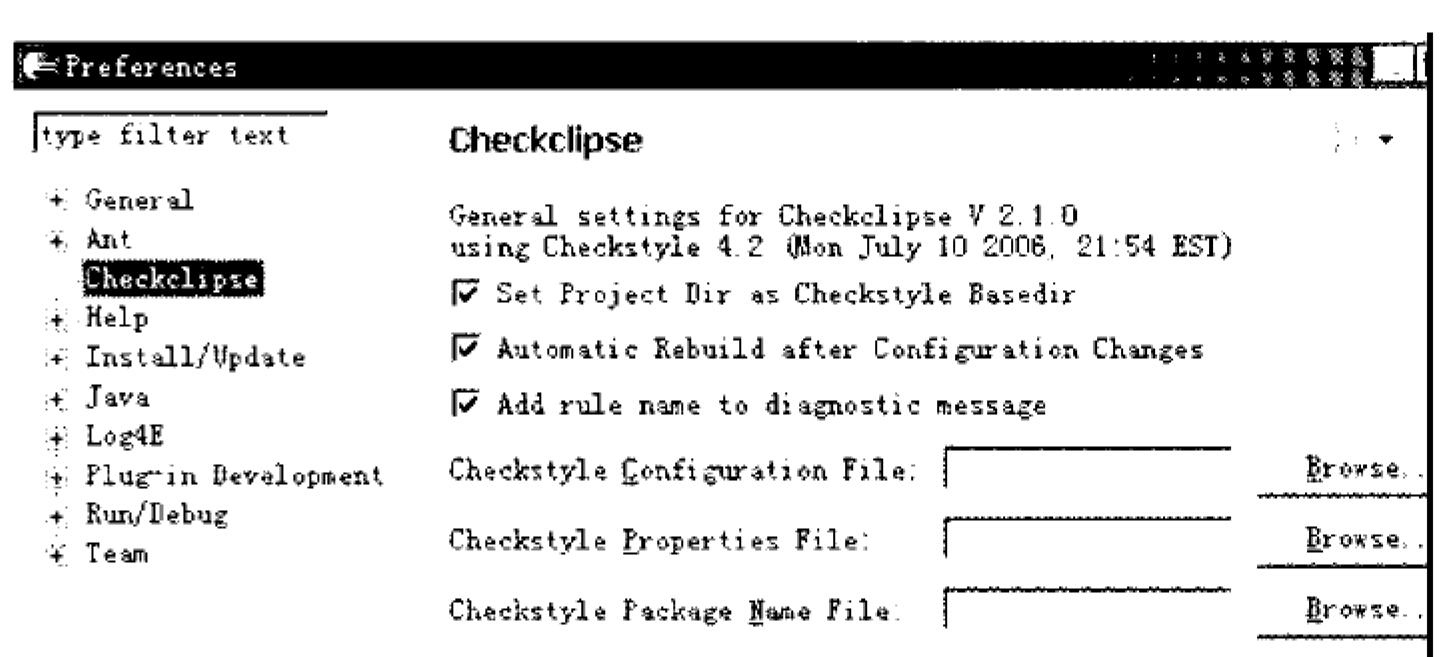


图 4-1　成功安装 Checkclipse 后的 Preferences 窗口

Checkclipse 有两个渠道可以进行配置，一个是全局的，一个是单个项目(Project)的。全局的可以在整个 Eclipse 的 workbench 中生效，而单个项目的配置可以在指定的项目中生效，它优先于全局的配置。

对于单个项目：右键单击某个项目，然后选择 Properties 命令就可以看到 Checkclipse 的窗口。在 Configuration 选项卡中，勾选 Enable Checkstyle 复选框，然后在 Checkstyle Configuration File 一行中选择 CheckStyle 配置文件就可以了，如图 4-2 所示。

对于全局的设置：选择 Window|Preferences 命令就可以看到。设置方法与单个项目的设置是一样的。

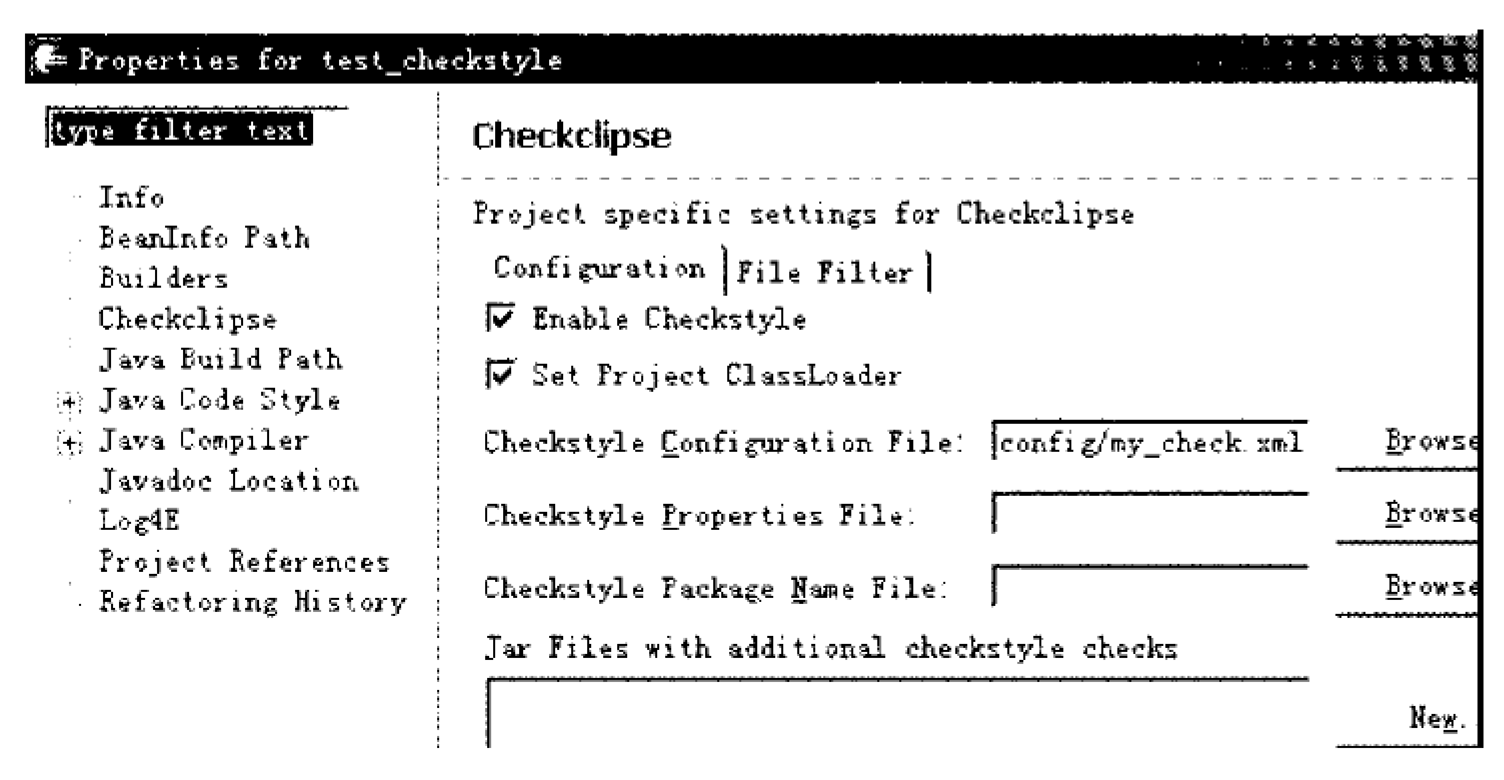


图 4-2　配置单个项目的 Checkclipse

经过上面的设置，Checkclipse 就可以使用了。如果想设置需要被检查的文件名，那么就在 File Filter 标签中修改被包含的文件。可以使用 Add、Remove、Change 等按钮进行编辑。Included Resources 中显示了被检查的文件清单，如图 4-3 所示。

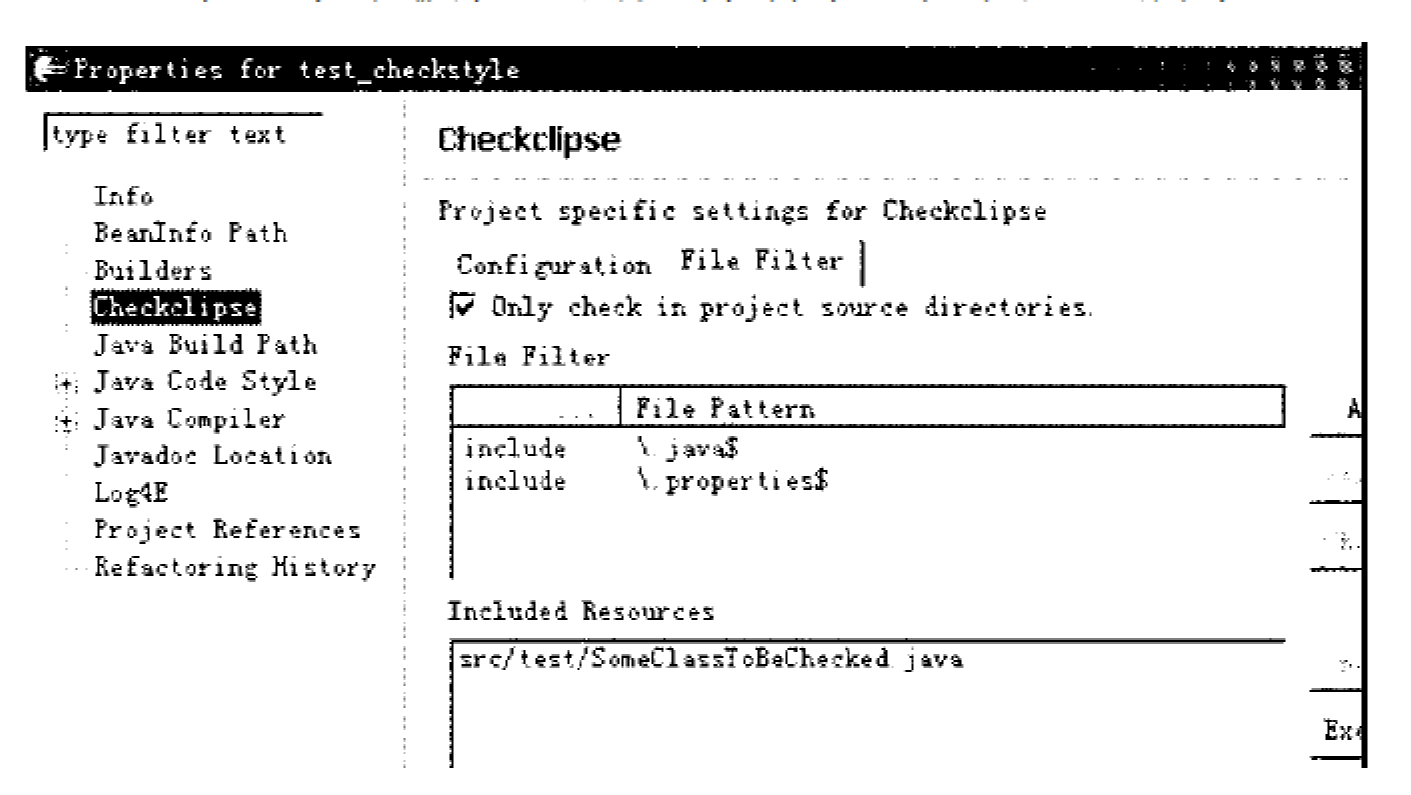


图 4-3　设置单个项目的 Checkclipse 的文件过滤器(file filter)

2. Checkclipse 的使用

(1) 建立一个 Eclipse 的项目 test_checkstyle。包含一个源文件夹 src，一个目标生成文件夹 eclipse_build，如图 4-4 所示。

```
test_checkstyle
  settings
  eclipse_build
  src
    test
      SomeClassToBeChecked.java
  .classpath
  .project
```

图 4-4　测试如何使用 CheckStyle 的项目

(2) 在项目中开启 CheckStyle：打开该 project 的属性，单击左侧的 Checkclipse 后，勾选 Enable Checkstyle 复选框，如图 4-2 所示。

(3) 建立一个测试用的 Class，比如 SomeClassToBeChecked，内容如下。

```
/*
 * Copyright (c) 2001-2008 Beijing BidLink Info-Tech Co., Ltd.
```

```
* All rights reserved.
* Created on 2008-2-22
*
* $Id: learn_in_5_min.xml,v 1.3 2008/03/03 03:43:44 Administrator Exp $
*/
package test;
public class SomeClassToBeChecked {
}
```

(4) 用 CheckStyle 检查它：右击项目名，选择 Build Project 命令，会把 src 文件夹进行编译，把 class 文件放到 eclipse_build 中。结束之后，可以看到图 4-5 中的代码第 10 行处有一个大括号，把鼠标移上去就会出现提示“Missing a Javadoc comment.”。同时，在 Problems 窗口中也有提示，如图 4-6 所示。

```
SomeClassToBeChec...  ×   messages.xml   learn_in_5_min.xml
 1  /*
 2   * Copyright (c) 2001-2008 Beijing BidLink Info-Te
 3   * All rights reserved.
 4   * Created on 2008-2-22
 5   *
 6   * $Id$
 7   */
 8  package test;
 9
10  public class SomeClassToBeChecked {
11
12  }
```

图 4-5　代码窗口中的错误提示



图 4-6　Problems 窗口中的错误提示

(5) 修改代码：既然提示说缺少了 Javadoc 注释，就把它加上，如图 4-7 所示。

```
SomeClassToBeChec...  ×   messages.xml   learn_in_5_min.xml
 1  /*
 2   * Copyright (c) 2001-2008 Beijing BidLink Info-Te
 3   * All rights reserved.
 4   * Created on 2008-2-22
 5   *
 6   * $Id$
 7   */
 8  package test;
 9  /**
10   * 这是个测试类，为了测试如何使用CheckStyle而建立的。
11   * @author 申思维 shensiwei@sina.com
12   *
13   */
14  public class SomeClassToBeChecked {
15  
16  }
```

图 4-7　增加了 class 的注释后的效果图

然后重新编译，可以看出，Warning 没有了，检查通过。

由于 CheckStyle 自带的规则检查非常严格，一般的项目 Warning 非常多。通常可针对需要，定义自己的规则检查。

3. 自定义规则

CheckStyle 没有图形化的定制器，所以需要手工修改配置文件。比如代码需要符合下列规则。

长度方面：文件长度不超过 1500 行，每行不超过 120 个字，方法不超过 60 行。

命名方面：类名不能以小写字母开头，方法名不能以大写字母开头，常量不能有小写字母。

编码方面：不能用魔法数(Magic Number），if 最多嵌套三层。

那么，检查配置文件（如命名成 my_check.xml ）应该是这样的：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC
        "-//Puppy Crawl//DTD Check Configuration 1.2//EN"
        "http://www.puppycrawl.com/dtds/configuration_1_2.dtd">
<module name="Checker">
    <module name="TreeWalker">
        <!-- 长度方面的检查 -->
        <!-- 文件长度不超过 1500 行 -->
        <module name="FileLength">
            <property name="max" value="1500"/>
        </module>
        <!-- 每行不超过 120 个字-->
        <module name="LineLength">
            <property name="max" value="120"/>
        </module>
        <!-- 方法不超过 60 行 -->
        <module name="MethodLength">
            <property name="tokens" value="METHOD_DEF"/>
            <property name="max" value="60"/>
        </module>

        <!-- 命名方面的检查，它们都使用了 CheckStyle 默认的规则。 -->
        <!-- 类名(class 或 interface) 的检查 -->
        <module name="TypeName"/>
        <!-- 方法名的检查 -->
        <module name="MethodName"/>
        <!-- 常量名的检查 -->
        <module name="ConstantName"/>
```

```
            <!-- 编码方面的检查 -->
            <!-- 不能用魔法数 -->
            <module name="MagicNumber"/>
            <!-- if 最多嵌套三层 -->
            <module name="NestedIfDepth">
                <property name="max" value="3"/>
            </module>
        </module>
    </module>
```

可以看出，想增加一个检查，就是增加一个<module/>节点，然后具体写明节点内容。

让 CheckStyle 使用指定的检查配置文件：打开项目属性，在 Checkclipse 中的 CheckStyle Configuration File 一栏中选定配置文件，然后确定，如图 4-2 所示。

然后重新编译项目，就会发现，CheckStyle 的规则如我们所愿：只检查在文件中配置的几项。并且它们是以 Error 级别进行提示，而不是默认检查时出现的 Warning 级别。比如在一个方法中增加 4 层嵌套（共 5 个 if），并将方法名大写，就会出现如图 4-8 所示的结果。

```
 */
package test;

public class SomeClassToBeChecked {
    //类名前无javadoc，但是由于我们没有配置javadoc
    //的检查，所以没报错。

    public void TestNestedIf(){
    //该方法名被大写，所以报错
        if(true){
            if(true){
                if(true){
                    if(true){
                        //嵌套的第四个if，报错
                        if(true){    }
        }}}}
    }
}
```

```
Search  Console  History  Problems
2 errors, 0 warnings, 0 infos
Description                                  Resource                  Path
Errors (2 items)
    Name 'TestNestedIf' must match p   SomeClassToBeCheck...    test_...
    Nested if-else depth is 4 (max a   SomeClassToBeCheck...    test_...
```

图 4-8　定制配置的检查结果

可以看到，出现了两个 Error：方法名的“Name xx must match pattern…”和 if 嵌套的“Nested if-else depth is 4…”。把它们都改过来，即把方法名小写，if 循环嵌套三层，然后重新编译即可。

4.2　代码静态分析工具 FindBugs

FindBugs 是一个开源的静态代码分析工具，基于 LGPL 开源协议，是无须运行就能对代码进行分析的工具。它检查类或者 JAR 文件，将字节码（*.class、*.jar）与一组缺陷模式进行对比以发现可能的问题。

FindBugs 自带多种检测器：①60 余种 Bad practice；②80 余种 Correntness；③1 种 Internationalization；④12 种 Malicious codevulnerability；⑤27 种 Multithreaded correntness；⑥23 种 Performance；⑦43 种 Dodgy 等。目前新版本一直在增加，也可以人工进行配置。

FindBugs 具有以下一些特点。

(1) FindBugs 主要着眼于寻找代码中的缺陷，这就与其他类似工具有些区别了，直接操作类文件（class 文件）而不是源代码。

(2) FindBugs 可以通过命令行、各种构建工具（如 Ant、Maven 等）、独立的 Swing GUI 或是以 Eclipse 和 NetBeans IDE 插件的方式来运行。

(3) FindBugs 输出结果可以是 XML 形式的、文本形式的、HTML 格式的。

(4) 开发者可以通过多种方式来使用 FindBugs，最常见的是在新编写模块的代码分析以及对现有代码进行更大范围的分析。

(5) 与其他静态分析工具不同，FindBugs 不注重编程风格和编程格式，而注重检测真正的 Bug 及潜在的性能问题，尤其注意了尽可能抑制误检测(false positives)的发生。

4.2.1　FindBugs 环境建立

使用 FindBugs 2.0 至少需要 JRE/JDK 1.5 以上版本，FindBugs 是平台独立的，可以运行于 GNU/Linux、Windows、MacOS X 等平台上。

FindBugs 可以通过三种方法使用，可以通过 Ant 工具，通过 Ant 提供的 Swing 操作界面和作为 Eclipse 的一个插件来使用。该报告主要介绍 Eclipse 里集成 FindBugs 与具体实例项目进行测试，其余两个简单介绍安装方式。

1. 通过 Ant 工具安装

Ant 工具是一个 Java 自动执行工具，它是在 build.xml 里编写脚本然后执行。

使用前要求 Ant 程序已安装或在 Eclipse 中已导入。构建 build.xml 格式如下:

```
<project name="项目名称，可任意 "default="all">
<property name="FindBugs.home" value=".. /FindBugs-2.0.2" />
    <path id="FindBugs.path">
        <fileset dir=".. /FindBugs-2.0.2">
            <include name="**/*.jar" />
```

```
        </fileset>
    </path>
        <taskdef name="FindBugs"
            classname="edu.umd.cs.FindBugs.anttask.FindBugsTask"
            classpathref="FindBugs.path" />
        <!--  定义 FindBugs 的 home，FindBugs 的 task 要使用  -->
        <target name="FindBugs">
            <mkdir dir="target/FindBugs"/>
            <FindBugs home="${FindBugs.home}"
output="html"outputFile="target/findbugs/youtubeVideoCrawler-fb.html">
        <!--  以上定义 FindBugs 查找的类路径  -->
            <auxClasspath path="${FindBugs.home}/lib/FindBugs-ant.jar" />
            <auxClasspath>
                <fileset dir="lib" includes="*.jar" />
                </auxClasspath>
                <sourcePath path="src" />
                <class location="bin" />
        <!--  以上定义项目的.class 路径，一定注意准确性  -->
            </FindBugs>
        </target>
 </project>
```

安装运行的步骤如下。

(1) 构建 build.xml，如图 4-9 所示。

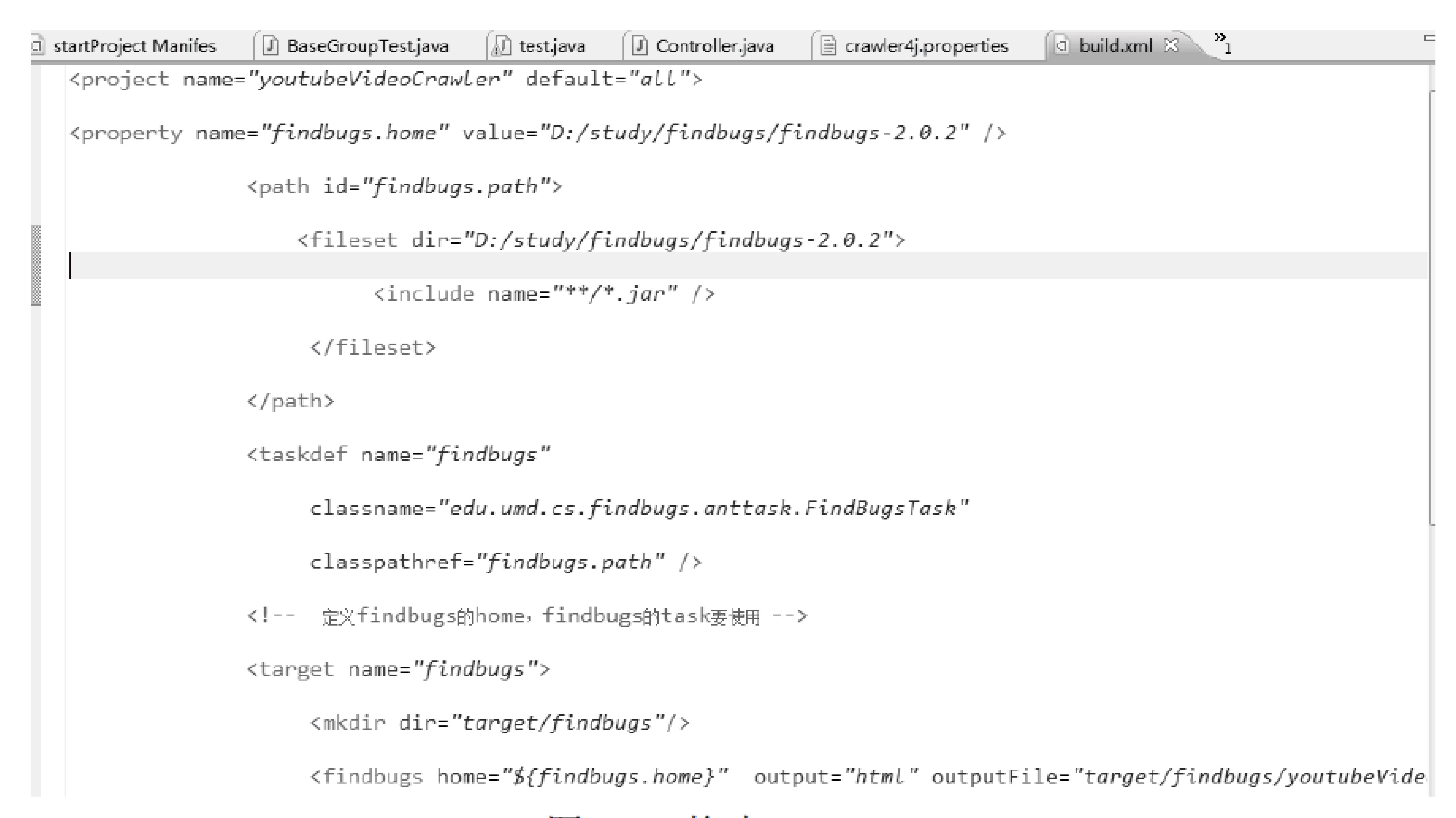


图 4-9　构建 build.xml

(2) 通过 Eclipse 运行 Ant build，然后启动 FindBugs，如图 4-10 所示。

(3) 在目的路径生成项目相关 FindBugs 报告。

2. 提供的 swing 工具

Ant 操作通过写 bulid.xml 完成。这对不熟悉 Ant 的人员来说操作起来会有些困难。

FindBugs 提供 Swing 工具。运行 FindBugs 解压包中 bin 文件夹下的 FindBugs.bat。（前提是已安装 Java 并且配置了 Java 环境变量。）

FindBugs 的 Swing 工具初始界面如图 4-11 所示。

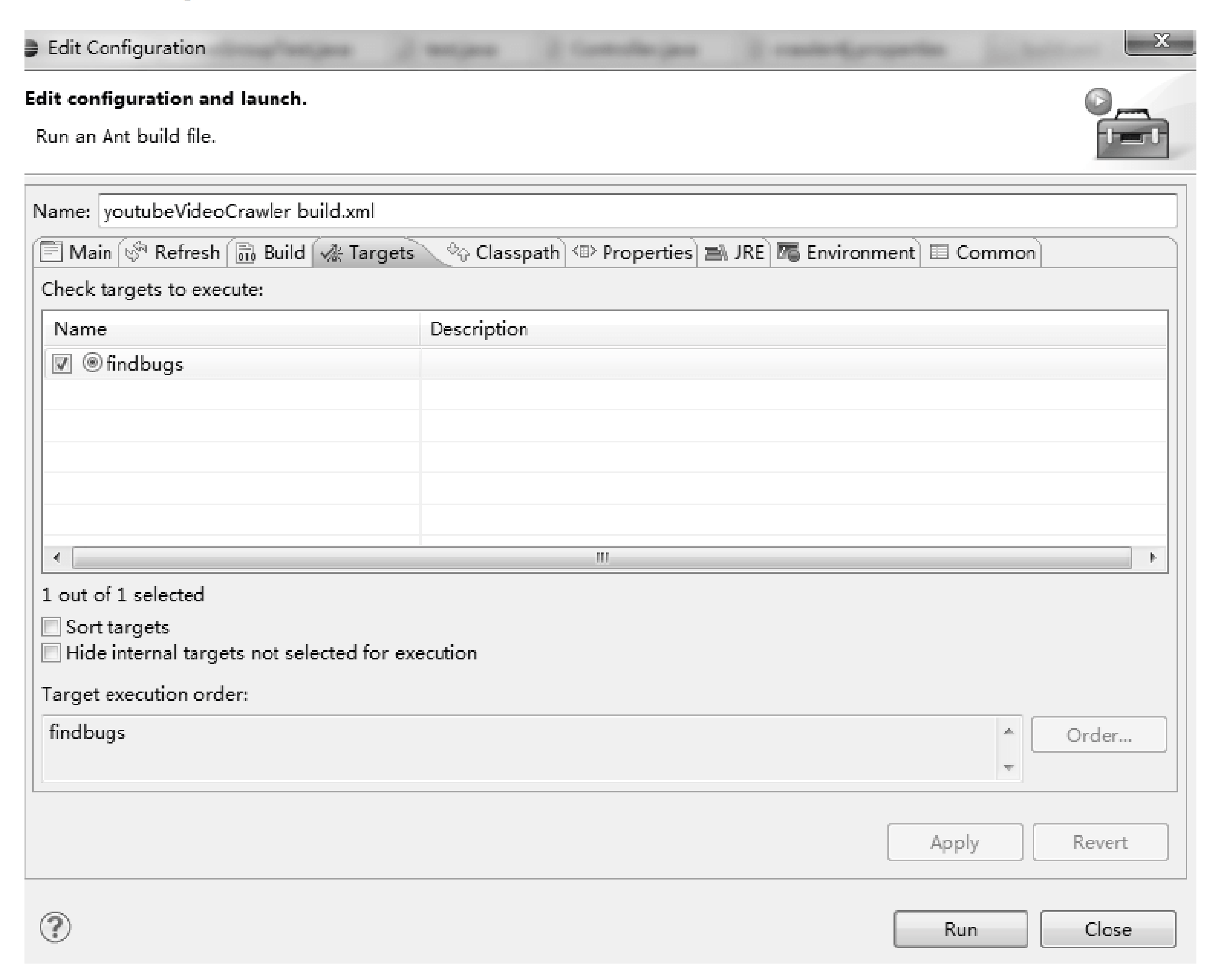


图 4-10　通过 Ant build 启动 FindBugs

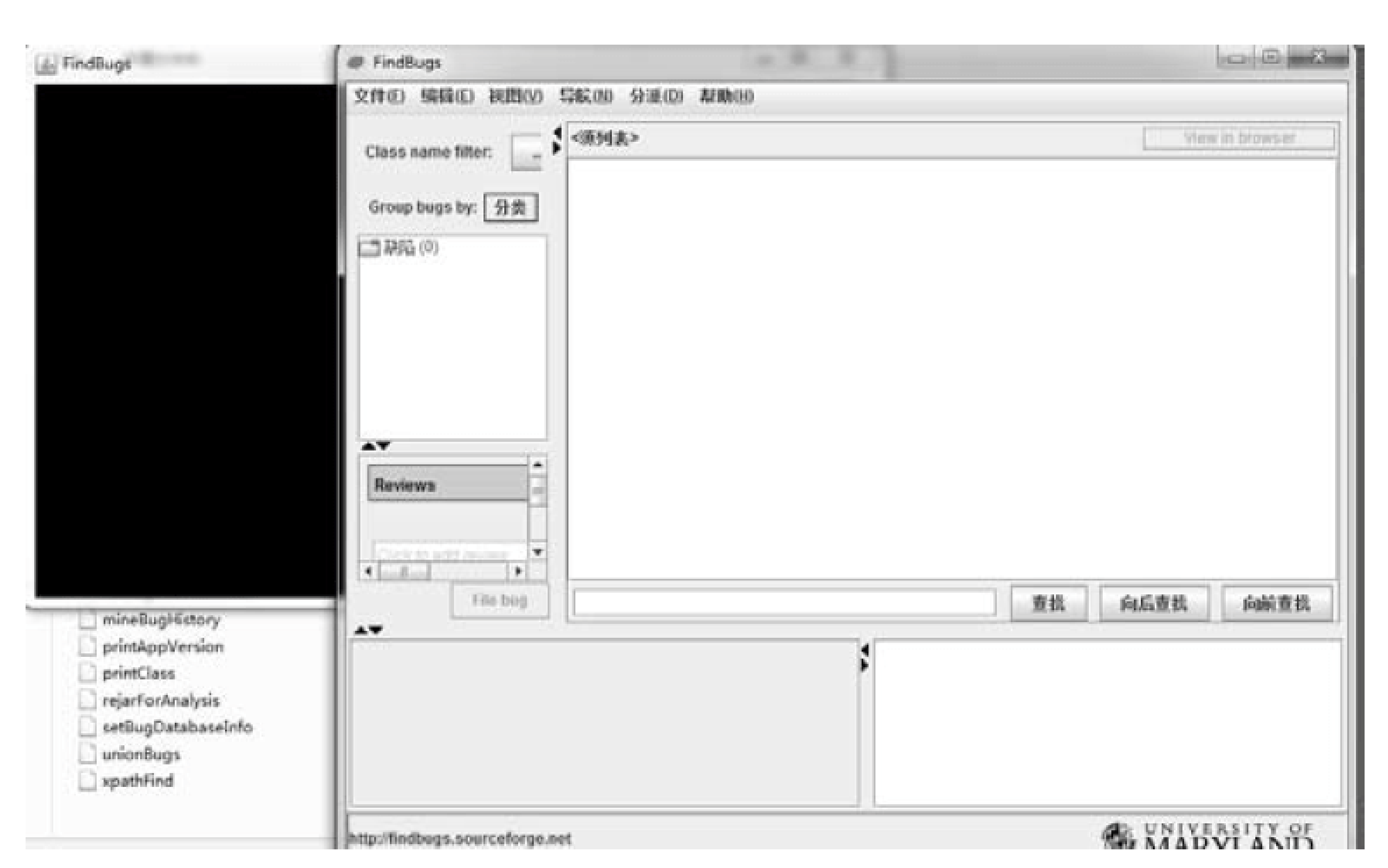


图 4-11　FindBugs 的 Swing 界面

可通过选择“文件”|“新建”命令，然后添加要分析的类包和目录（可以选择编译好的类所在的文件夹，也可以选择生成的 jar 包），再添加辅助类所在的文件夹和源文件所在的文件夹（Java 文件所在的文件夹）。再单击“完成”按钮就可以建立一个要分析的项目，如图 4-12 所示。

建立项目后，会自动开始解析项目。解析后的界面——即分析 Bug 界面，如图 4-13 所示。其中左边是缺陷的树结构列表，单击其中一个 Bug，可以在右边的界面中显示 Bug 的源文件以及所在的位置。

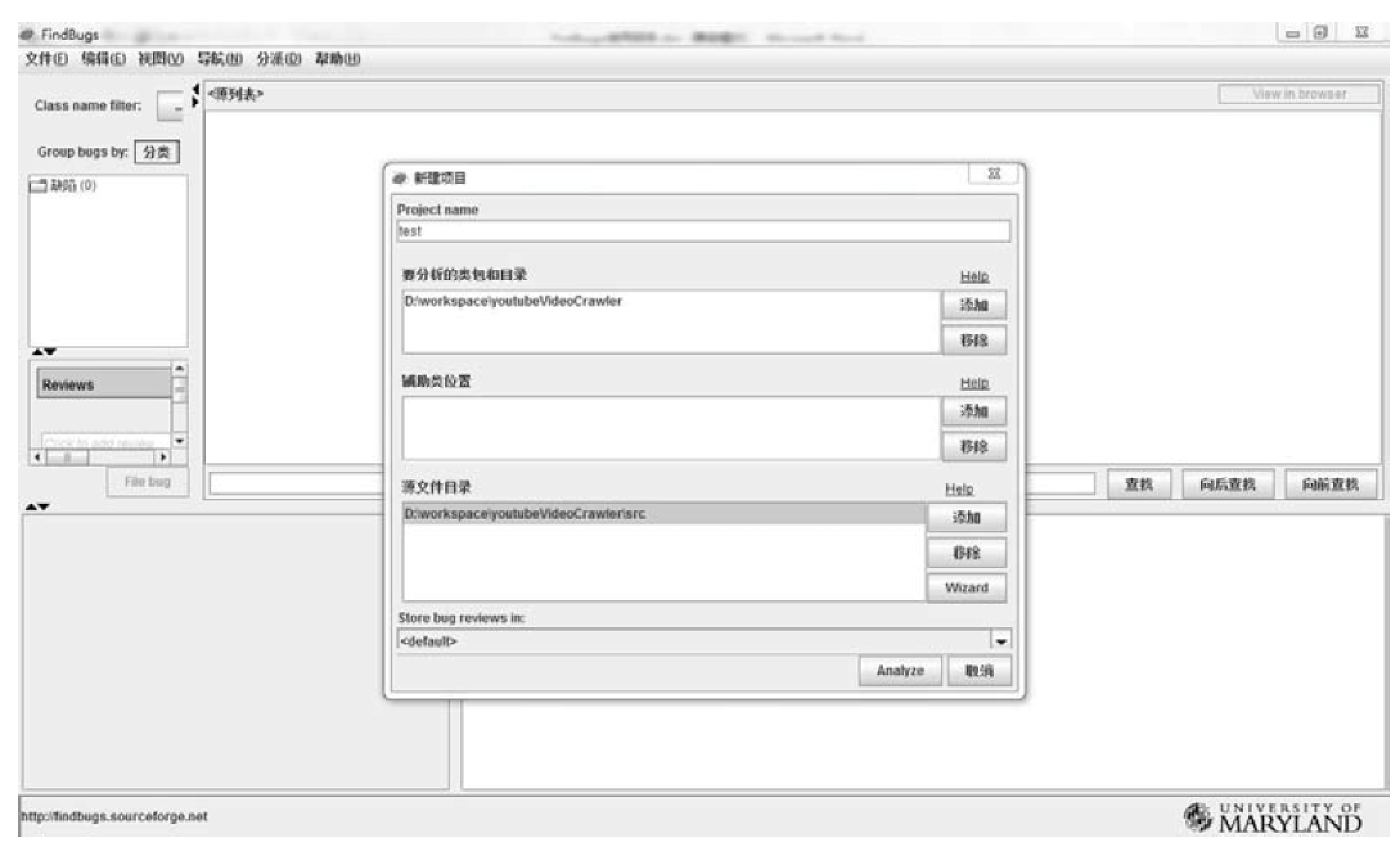


图 4-12　FindBugs 分析项目图

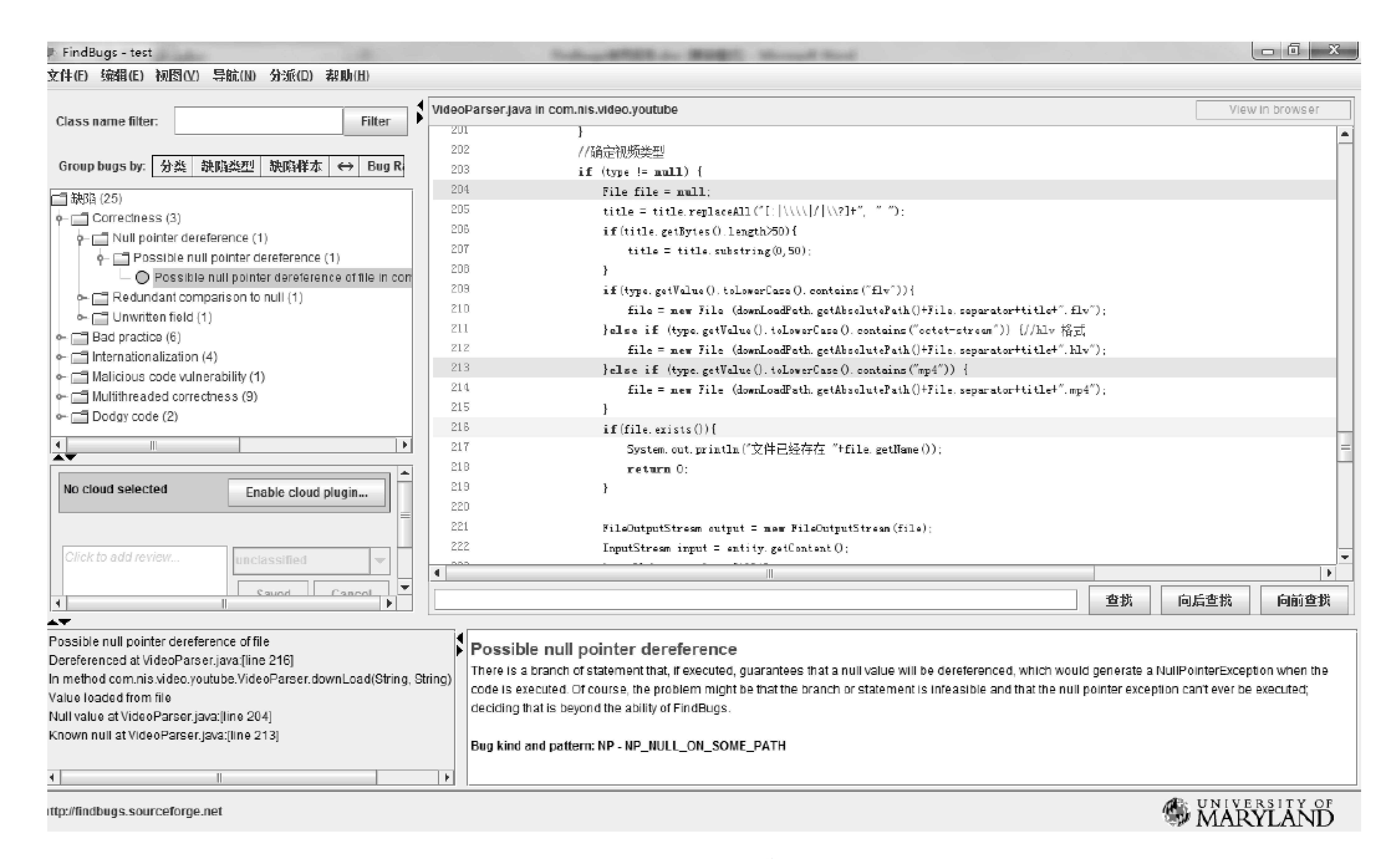


图 4-13　Bug 分析界面

3. FindBugs Eclipse 插件安装

Eclipse 的 FindBugs 插件，可以将 FindBugs 集成到 Eclipse 中使用。安装需要 Eclipse 版本 3.3 及以上，并且 JRE/JDK 1.5 及以上。目前可有以下两种安装方式。

1) 在线安装

(1) 使用 Eclipse 的 Help|Install New Software…|Add respository 以插件形式在线安装

FindBugs，如图 4-14 所示。

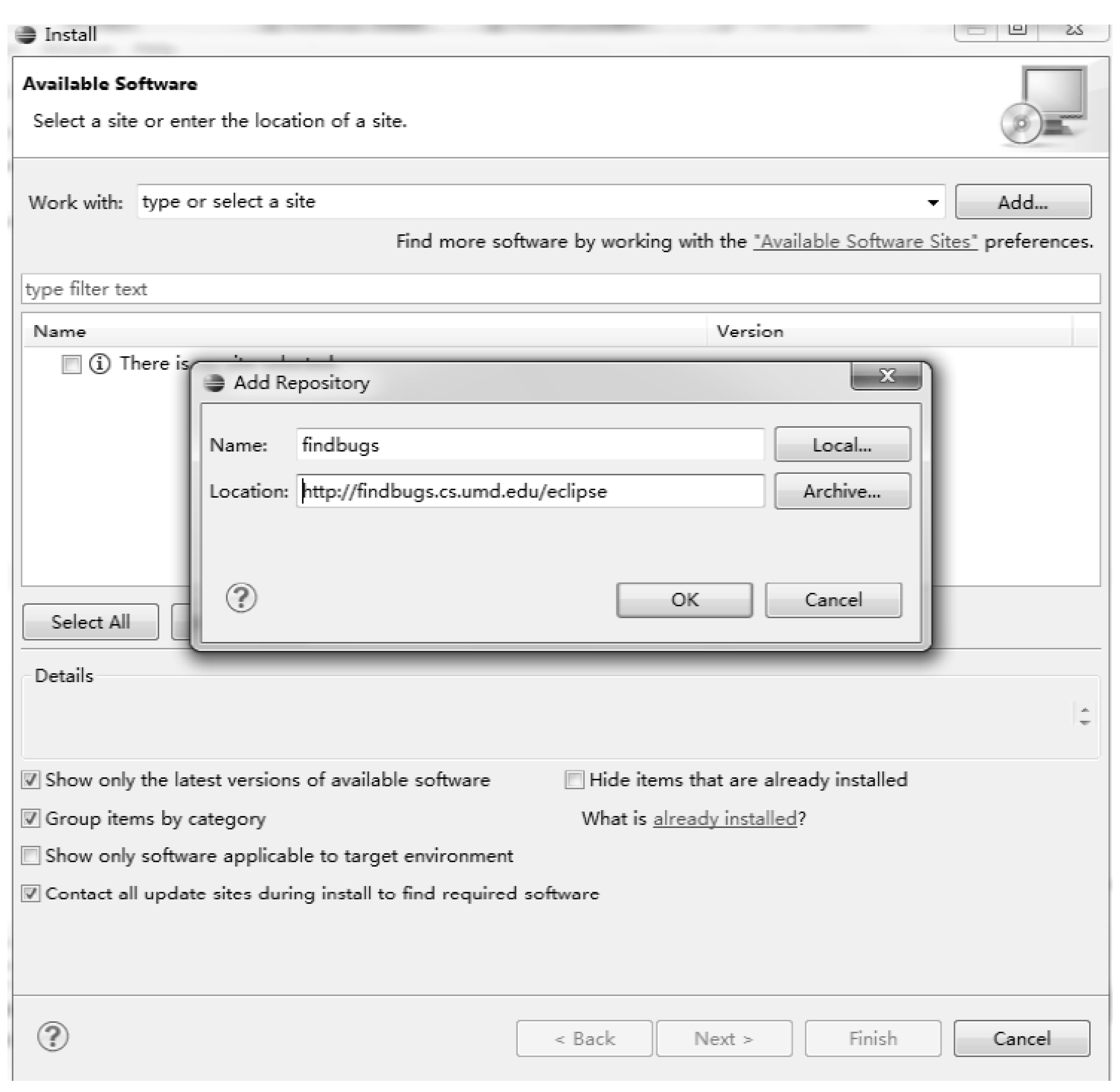


图 4-14　输入 FindBugs 下载路径

(2) 单击 OK 按钮，然后选择所有安装选项，单击 Next 按钮，查找相关依赖项，如图 4-15 所示。

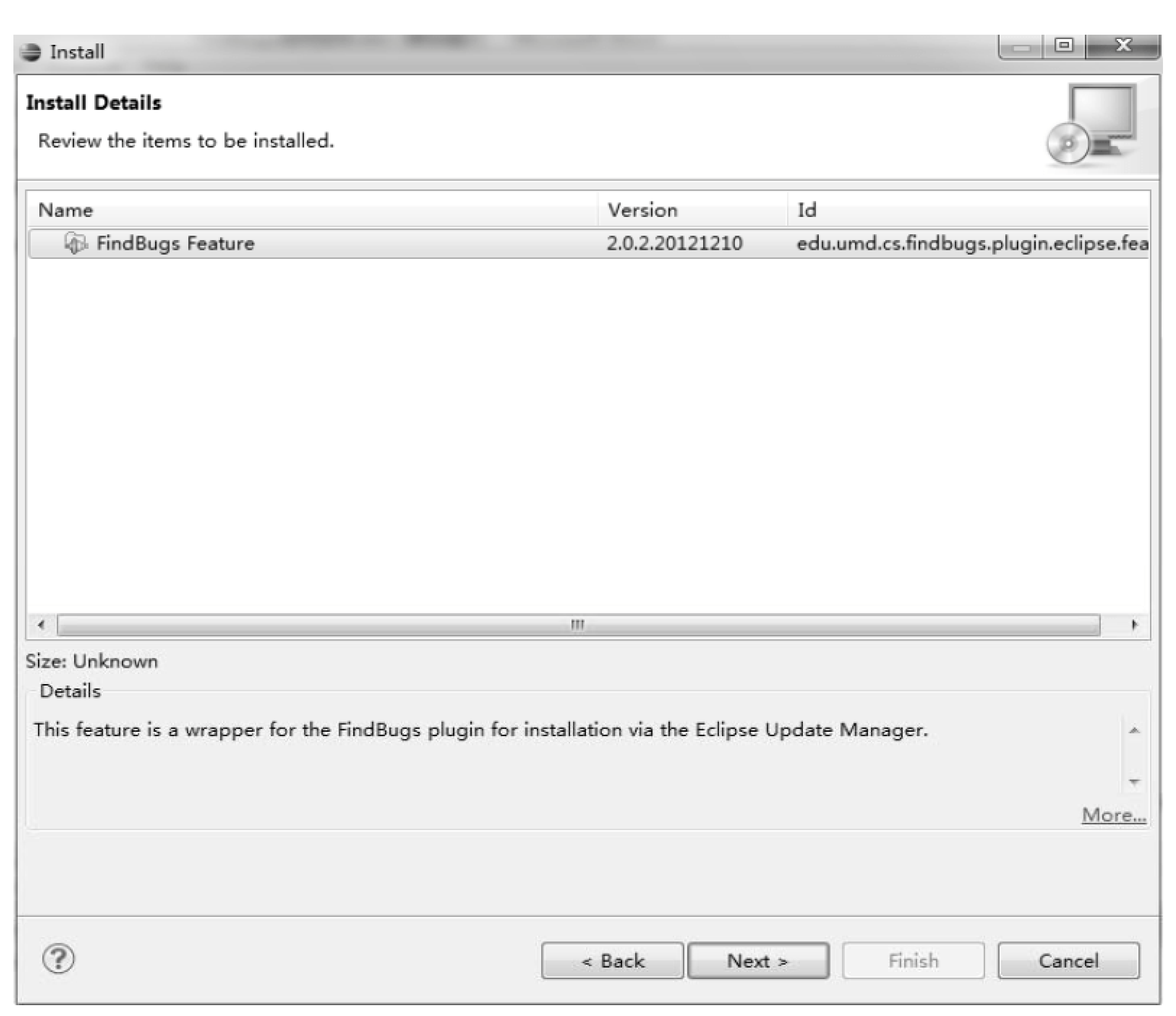


图 4-15　选择 FindBugs 相关安装选项

(3) 接受相关协议，完成插件安装。单击 Finish 按钮完成安装，如图 4-16 所示。

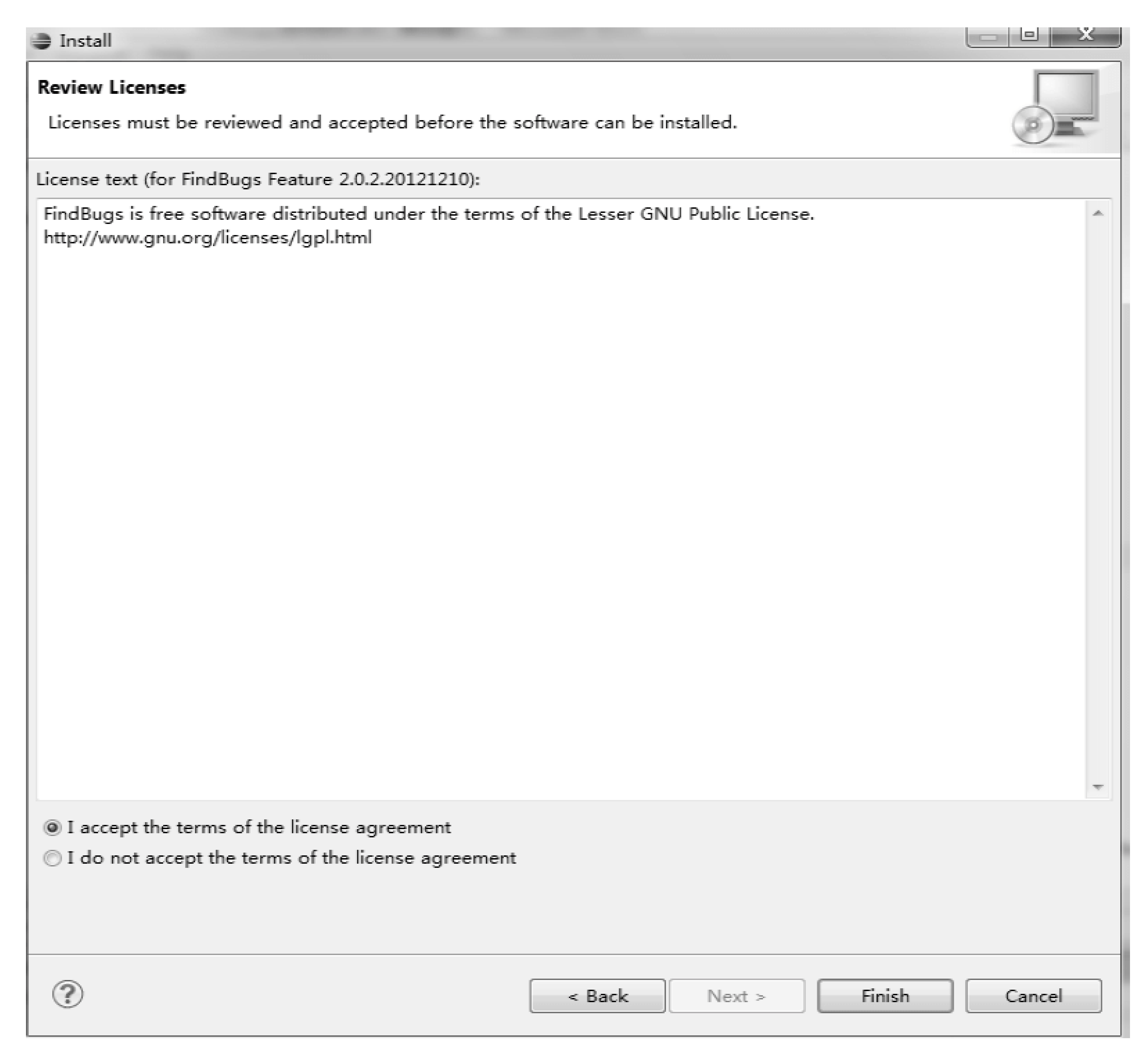


图 4-16　结束安装

2) 离线安装

(1) 到网站下载最新版本 FindBugs，目前版本为 3.0.0。将下载的压缩文件解压到 Eclipse 的 plugins 子目录中，重新启动 Eclipse，如图 4-17 所示。http://sourceforge/progects/findbugs。

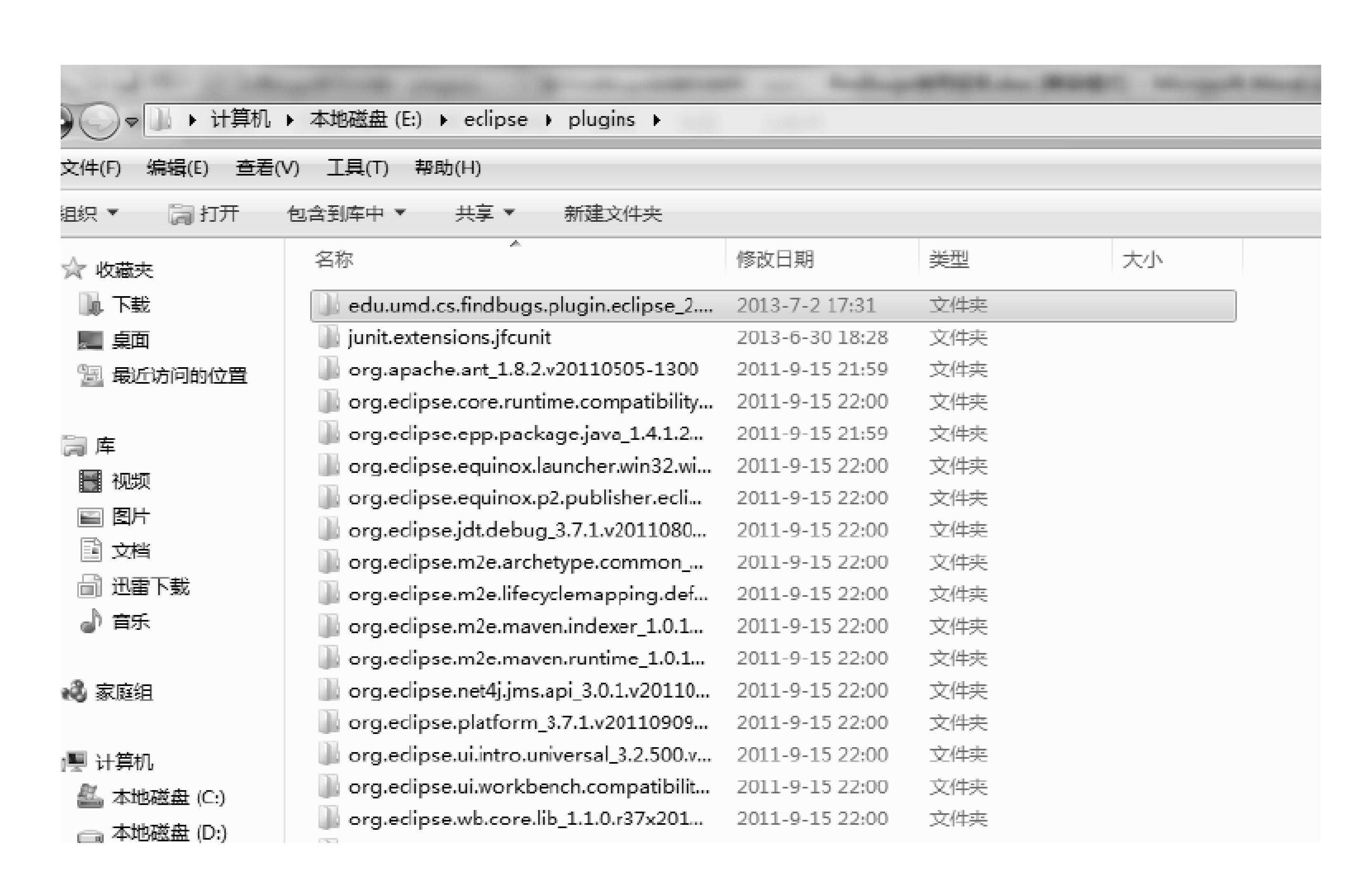


图 4-17　FindBugs 插件放入 Eclipse 下

(2) 打开 Eclipse Help|About Eclipse Platform|Plug-In Details 可以查看 FindBugs 的版本信息；当然，如果没有正确安装不会显示 FindBugs 信息。安装成功后如图 4-18 所示。

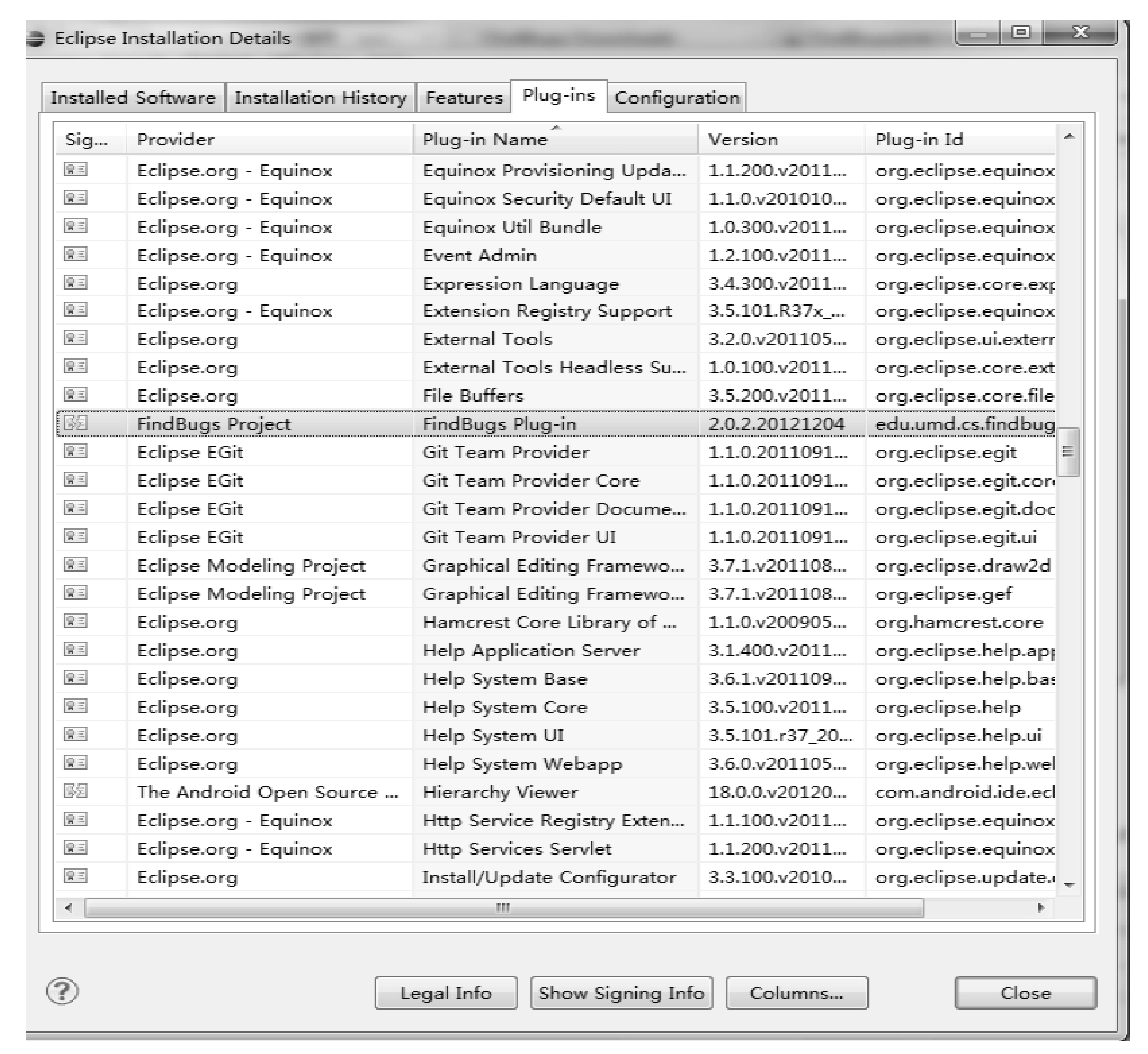


图 4-18　Eclipse 安装 FindBugs 插件成功

4. 插件简单测试

新建一个测试项目 test，写一个有问题的程序。右击源文件，选择 Find Bugs|Find Bugs 命令，如图 4-19 所示。

```
package com.topsoft.findbugs;
public class FindBugsTest {
    private String[] name;
    public String[] getName() {
        return name;
    }
    public void setName(String[] name) {
        this.name = name;
    }
}
```

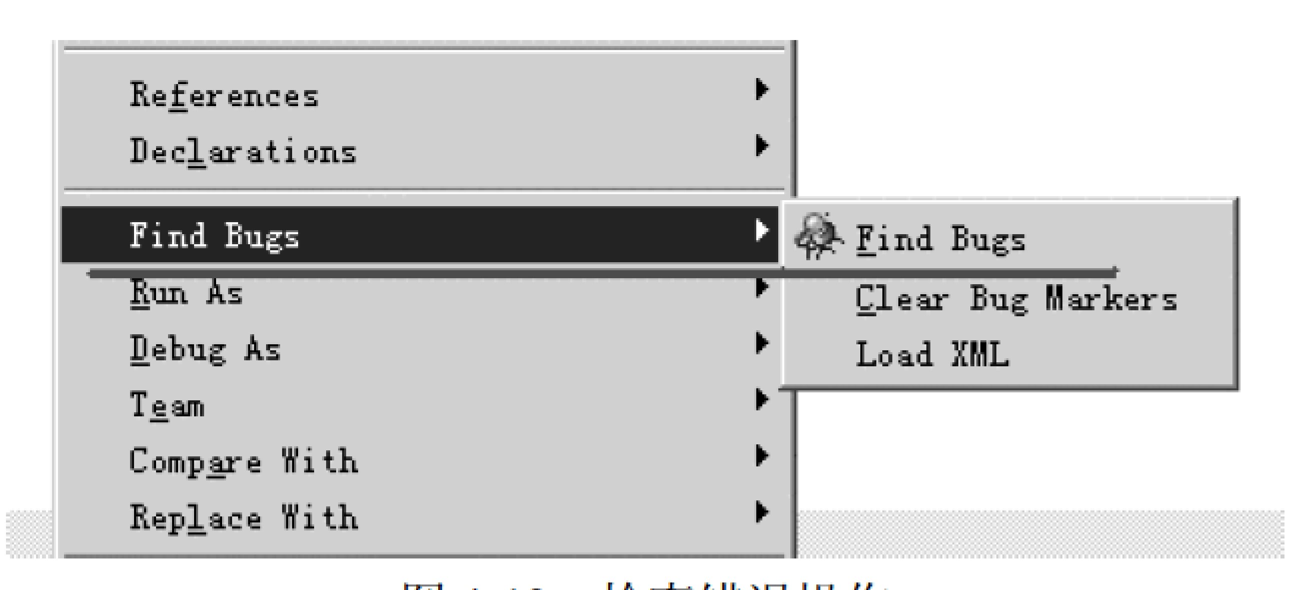


图 4-19　检查错误操作

此时源代码左边出现两个 Bug 图标，如图 4-20 所示。

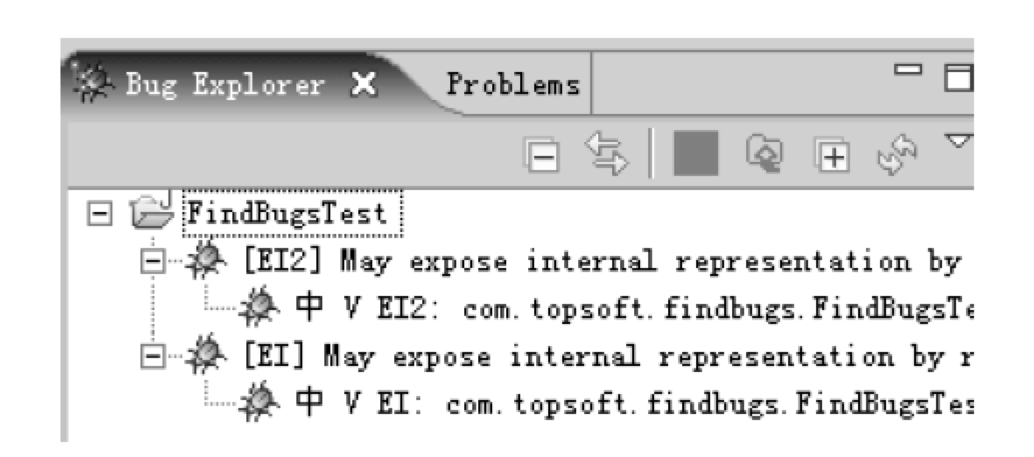


图 4-20　检查出的错误显示

可以看到黄色虫子。找出的 Bug 虫子颜色有三种：黑色的臭虫标志是分类，红色的臭虫标志表示严重 Bug 发现后必须修改代码，橘黄色的臭虫标志表示潜在警告性 Bug 尽量修改。

Bugs Explore 右侧 Problems 是问题信息，打开 Problems 面板，如图 4-21 所示显示错误原因。然后，根据提示进行代码修改。

Bug Details | Bookmarks | Tasks | Progress | JUnit | Problems

0 errors, 2 warnings, 0 infos

Description	Resource	Path	Location
Warnings (2 items)			
中 V EI: com.topsoft.findbugs.FindBu	FindBugsTest....	FindBugsTest/src/com/tops...	line 7
中 V EI2: com.topsoft.findbugs.FindB	FindBugsTest....	FindBugsTest/src/com/tops...	line 11

图 4-21　错误原因显示

选中相应的问题条目，右击，在弹出的菜单中，可以看到 Show Bug Details，选中也可以查看问题详细信息，选中第一个问题描述如图 4-22 所示。

根据这里详细的信息，可以得到 FindBugs 为什么会对代码报警告信息，及相应的处理办法，根据它的提示可以快速方便地进行代码修改。

如果双击问题，系统会自动跳转到相对应的问题行处。

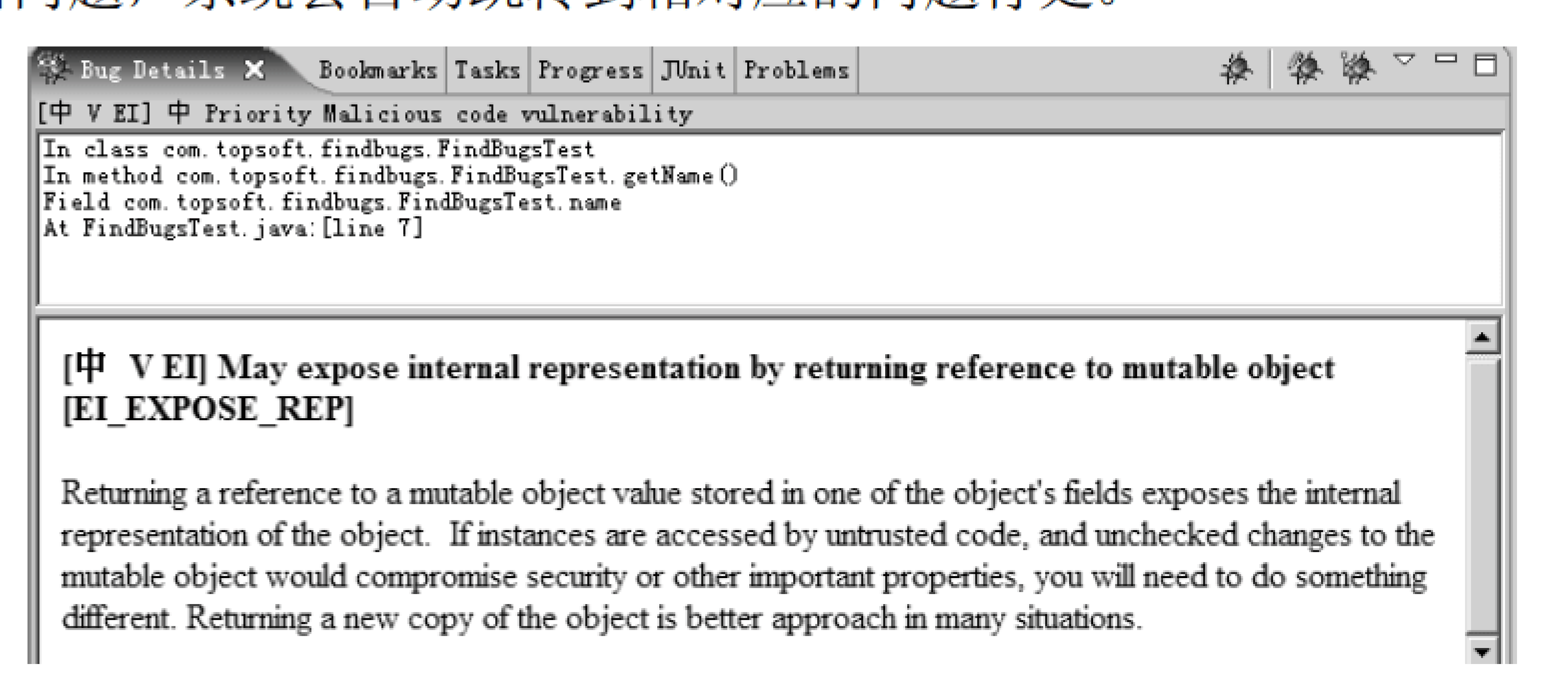


图 4-22　错误详细描述

5. FindBugs 的使用流程

FindBugs 的使用流程概括起来可以分为以下几步。

1) 安装

安装过程和通常的插件安装没有什么不同。安装结束后可以对其进行相应的配置。

选择项目，右击，选择 Properties|FindBugs，如图 4-23 所示。

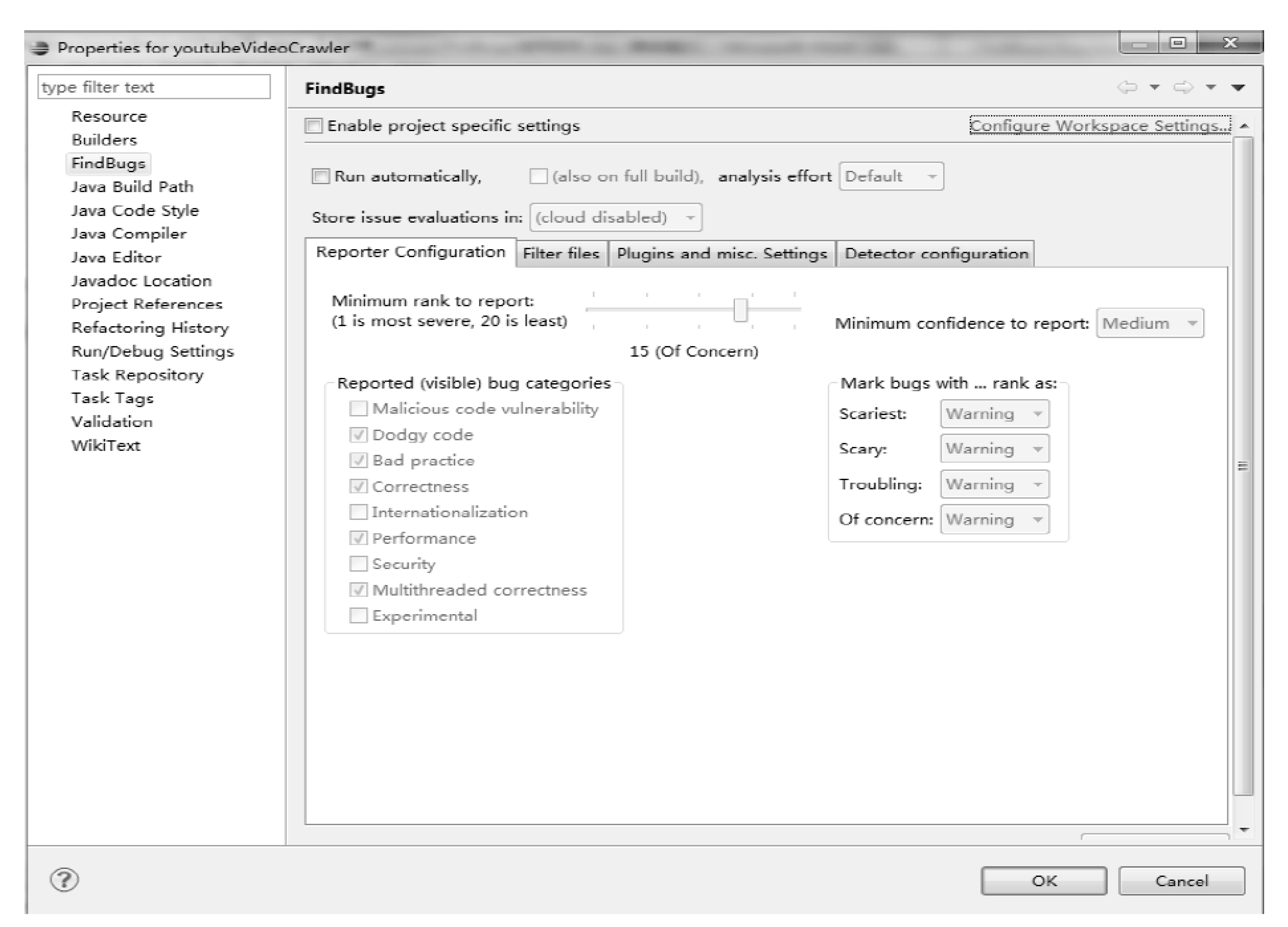


图 4-23　FindBugs 具体属性配置界面

2) 配置

可以配置的信息包括如图 4-23 所示相关设置，各项设置简要解释如下。

(1) Enable project specific settings

选择该选项后，会针对该项目进行特殊设置。

(2) Run automatically 开关

设置 Eclipse 自动编译开关——即主窗口菜单 Project|Build Automatically 这个选项勾上就行了。

当选中此项后，FindBugs 将会在修改 Java 类时自动运行，如果设置了 Eclipse 自动编译开关后，当修改完 Java 文件保存，FindBugs 就会运行，并将相应的信息显示出来。当此项没有被选中，则只能每次在需要的时候自己去运行 FindBugs 来检查代码。

(3) Detector Configuration 选择项

这里可以选择所要进行检查的相关的 Bug Pattern 条目，可以根据需要选择或去掉相应的检查条件，如图 4-24 所示。

(4) Reporter Configuration 选择项

① Minimum priority to report 选择项。

这个选择项是让用户选择哪个级别的信息进行显示，有 Low、Medium、High 三个选择项可以选择，类似于 Log4J 的级别设置。比如，选择了 High 选择项，那么只有是 High 级别的提示信息才会被显示。若选择了 Medium 选择项，那么只有是 Medium 和 High 级别的提示信息才会被显示。若选择了 Low 选择项，那么所有级别的提示信息都会被显示。

② Report bug categories 选择项。

在这里是一些显示 Bug 分类的选择。

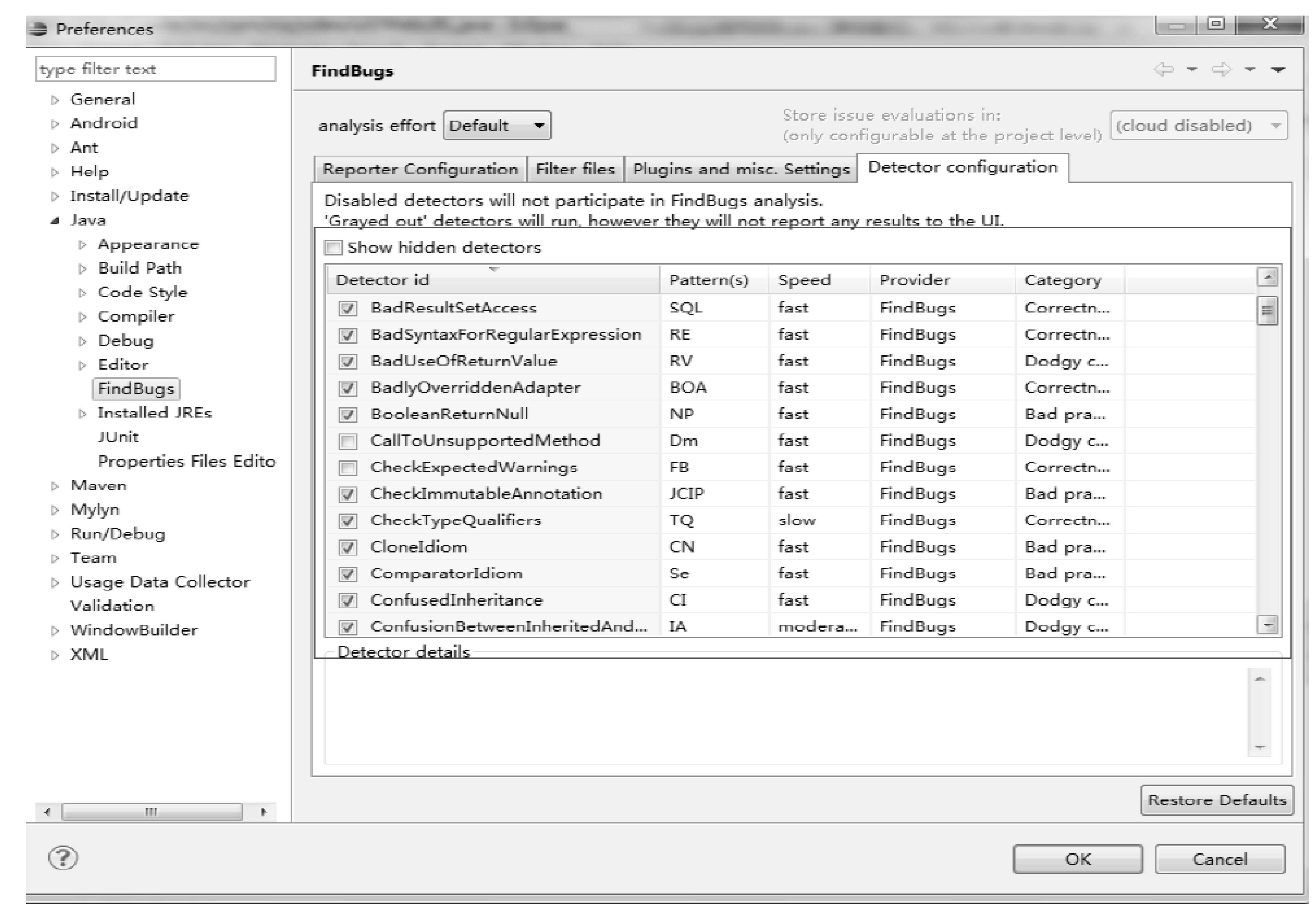


图 4-24　检测器配置

- Malicious code vulnerability：关于恶意破坏代码相关方面的。
- Dodgy code：关于该类型下的问题代码导致高危 Bug 的可能性很高。
- Bad practice：与最佳实践相反，这种类别下的代码违反了公认的最佳实践标准，比如某个类实现了 equals 方法但未实现 hashCode 方法等。
- Correctness：关于代码正确性相关方面的。
- Internationalization：关于代码国际化相关方面的。
- Performance：关于代码性能相关方面的。
- Security：关于代码安全性防护。
- Multithreaded correctness：关于代码多线程正确性相关方面的。
- Experimental：关于实验性问题。
- Style：关于代码样式相关方面的。

4.2.2　FindBugs 应用举例

与其他静态分析工具不同，FindBugs 不注重编程风格或者格式，而是使用内置检查器去寻找真正的缺陷或者潜在的性能问题。FindBugs 也可以找到代码中的一些真正问题，帮助用户提高代码质量与消除隐患，尤其它有丰富的 Bug 描述提示问题该如何解决。下面就用一个简单的“视频下载与抓取系统”作为例子加以说明。

1. 导入项目

打开 Eclipse，选择 File|Import…命令，在 Import 窗口中选择 Existing Projects into

Workspace，如图 4-25 所示。

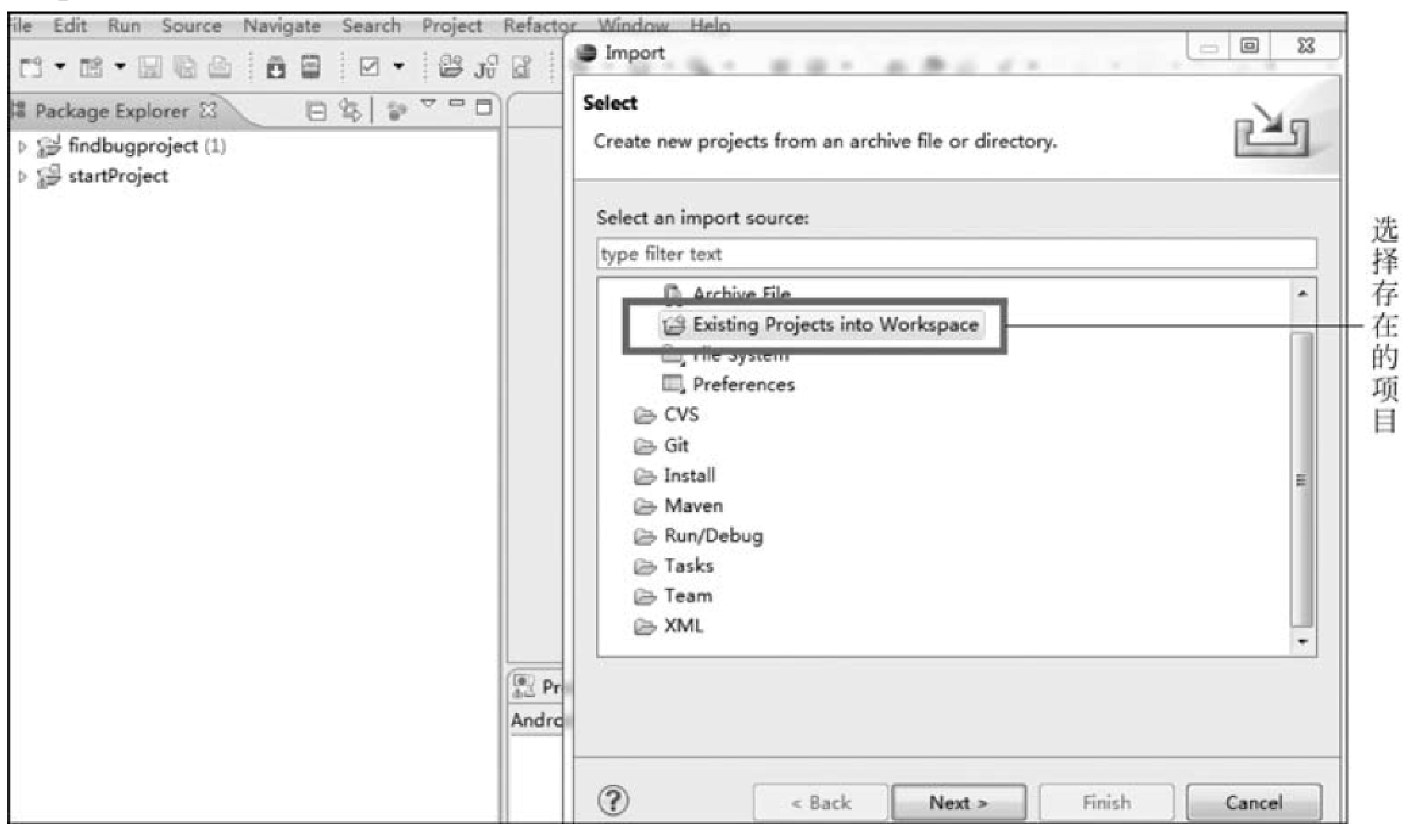


图 4-25　选择项目

双击进入 Import 窗口，浏览选择要导入的项目，单击 Finish 按钮完成导入，如图 4-26 所示。

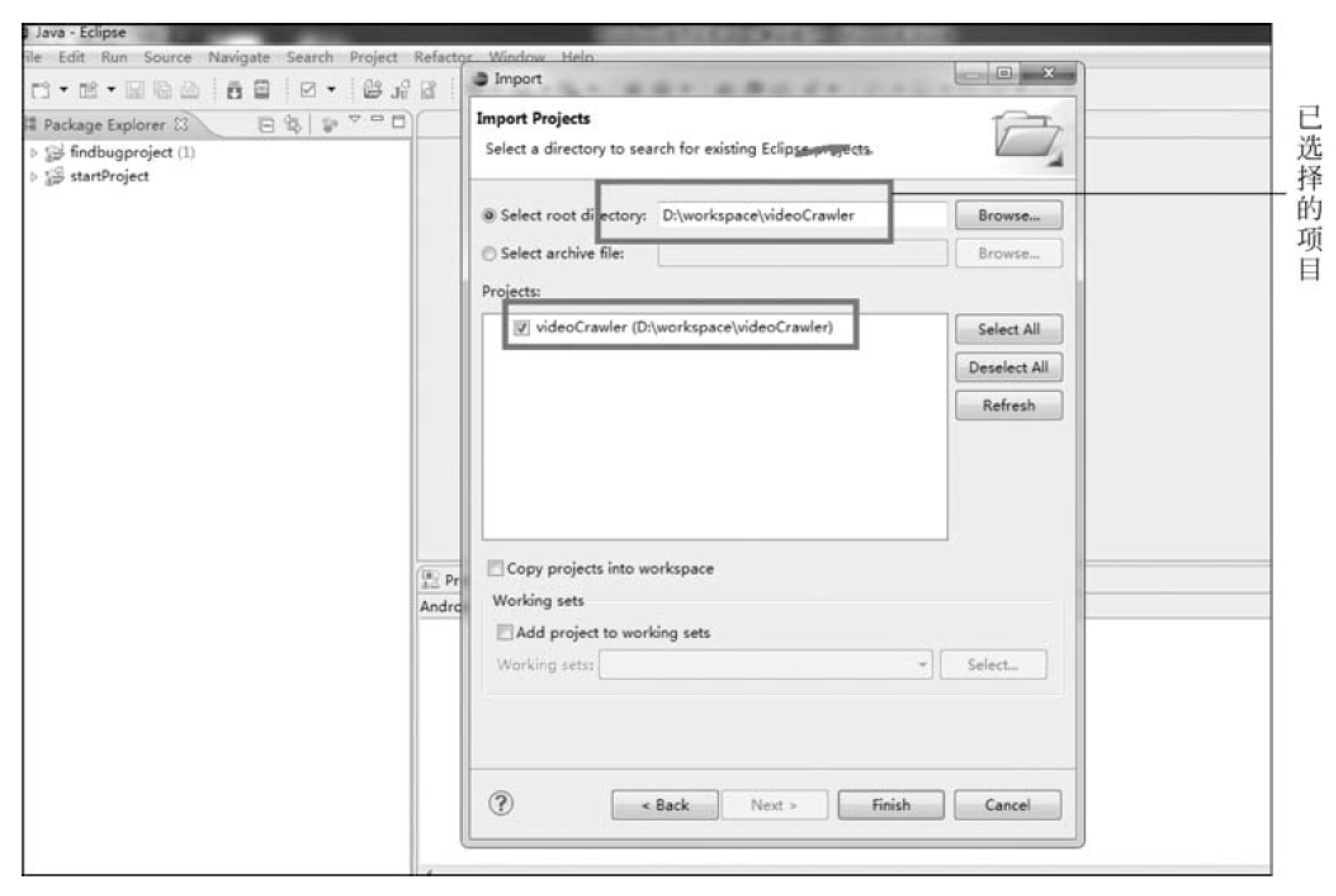


图 4-26　导入项目

2. 配置项目

完成导入后，在该项目上右击选择 Properties|FindBugs|Enable project specific settings，然后进行特殊设置，如图 4-27 所示。

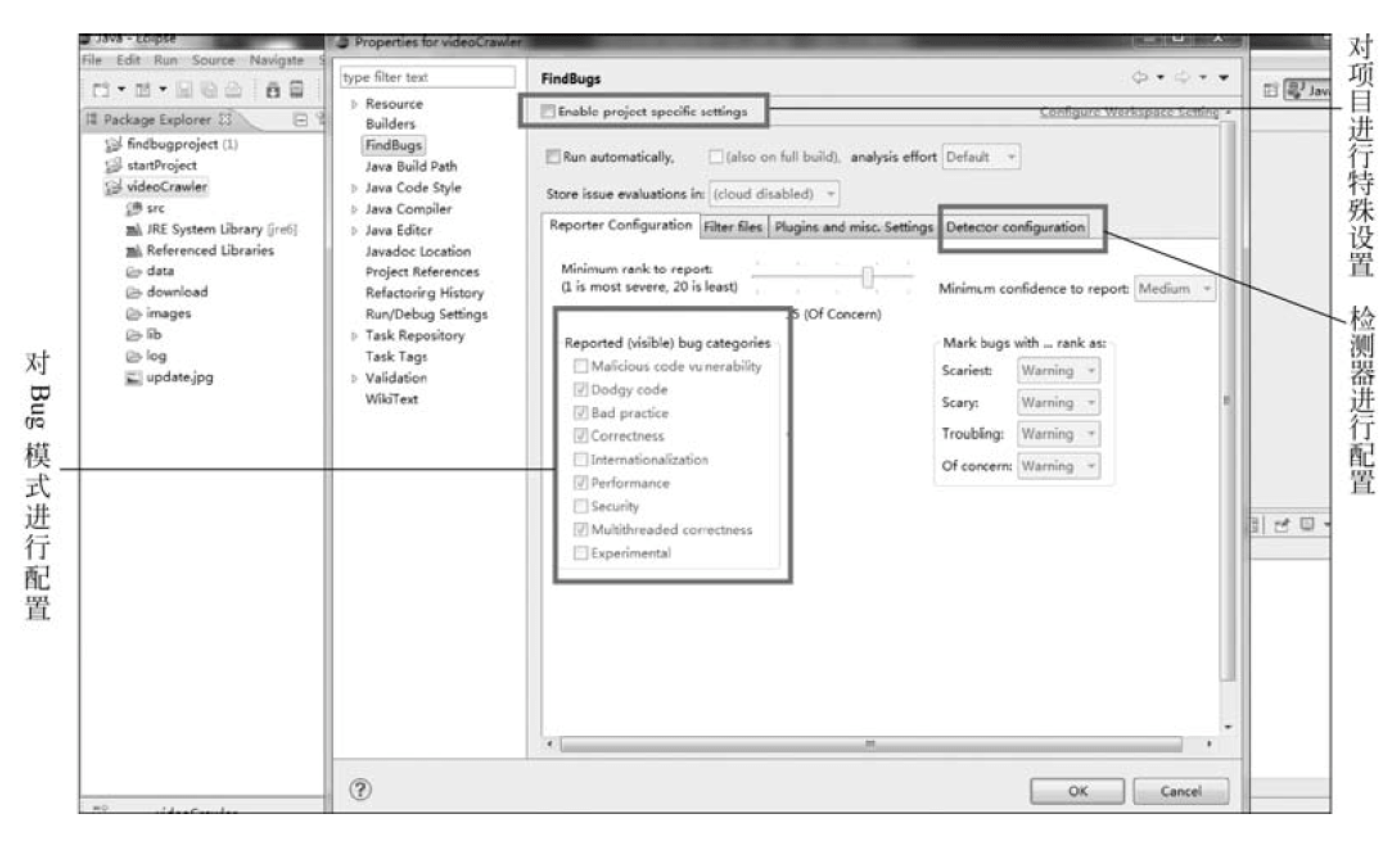


图 4-27　项目进行 FindBugs 特殊配置

在 Reporter Configuration 选项卡中选择 Security、Experimental 复选框。单击 OK 按钮确认完成，如图 4-28 所示。

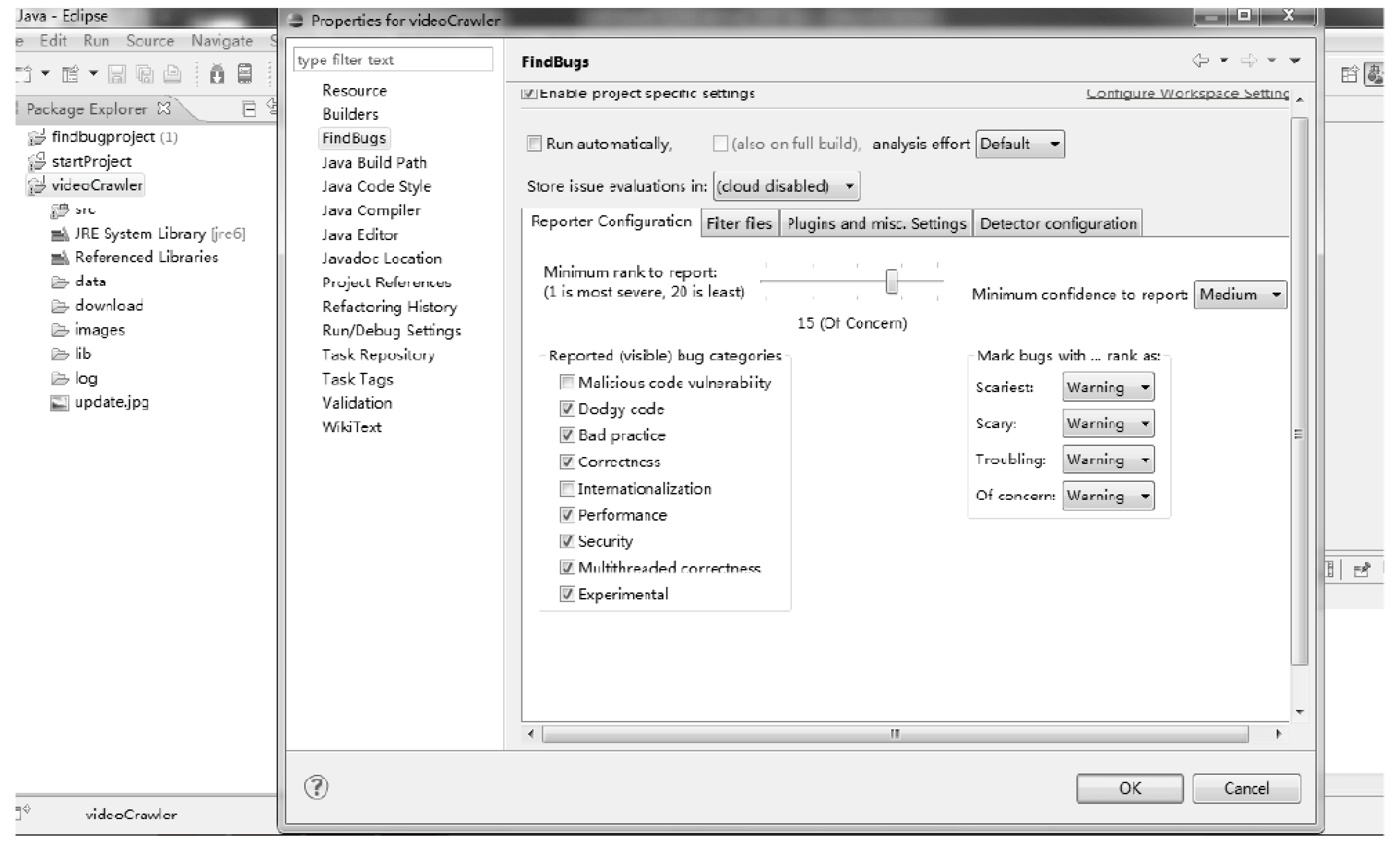


图 4-28　FindBugs 增加配置结果

3. 测试并处理错误

右击 videoCrawler 项目，选择 Find Bugs|Find Bugs，进行项目 Bug 分析，分析完成后在项目后边显示 Bug 数（该插件只分析 Java 类，其他不做分析）。如图 4-29 所示有 20 个 Bug。然后分别进行处理。

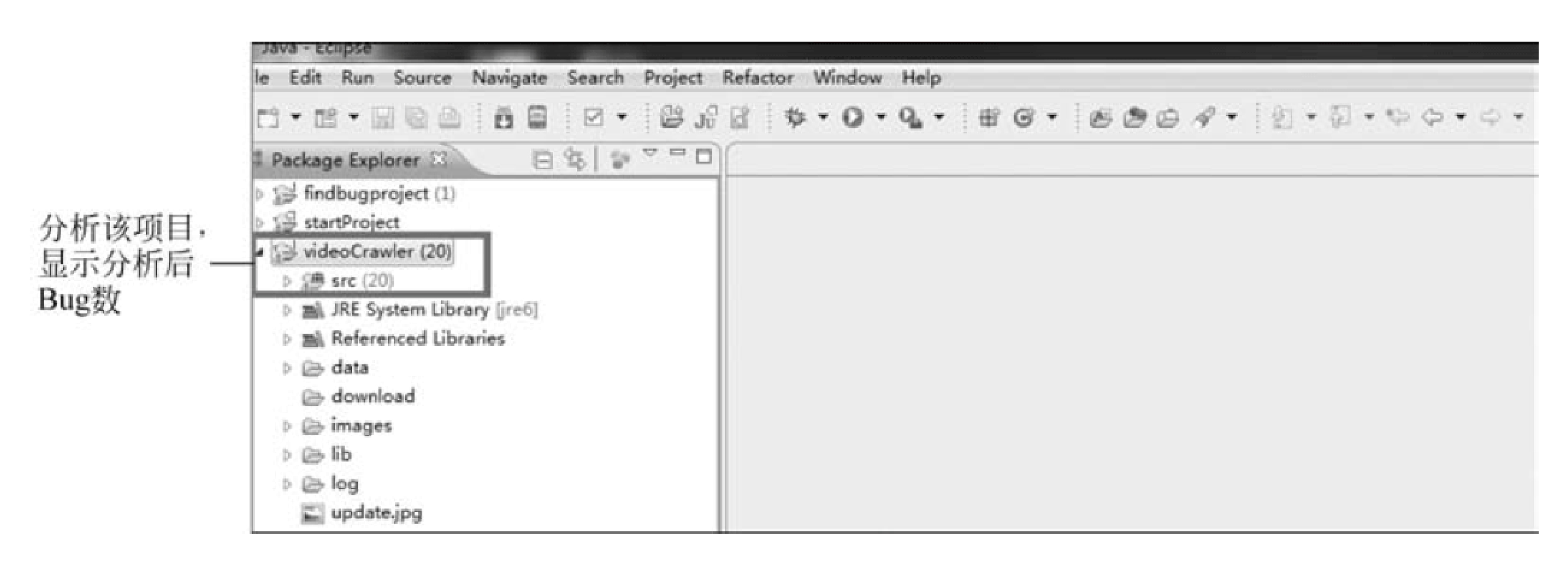


图 4-29　FindBugs 分析后显示 Bug 数

打开 FindBugs 视图 Bug Explorer，可看到该项目的所有 Bug，然后对每个 Bug 进行具体分析和有针对性地处理，如图 4-30 和图 4-31 所示。

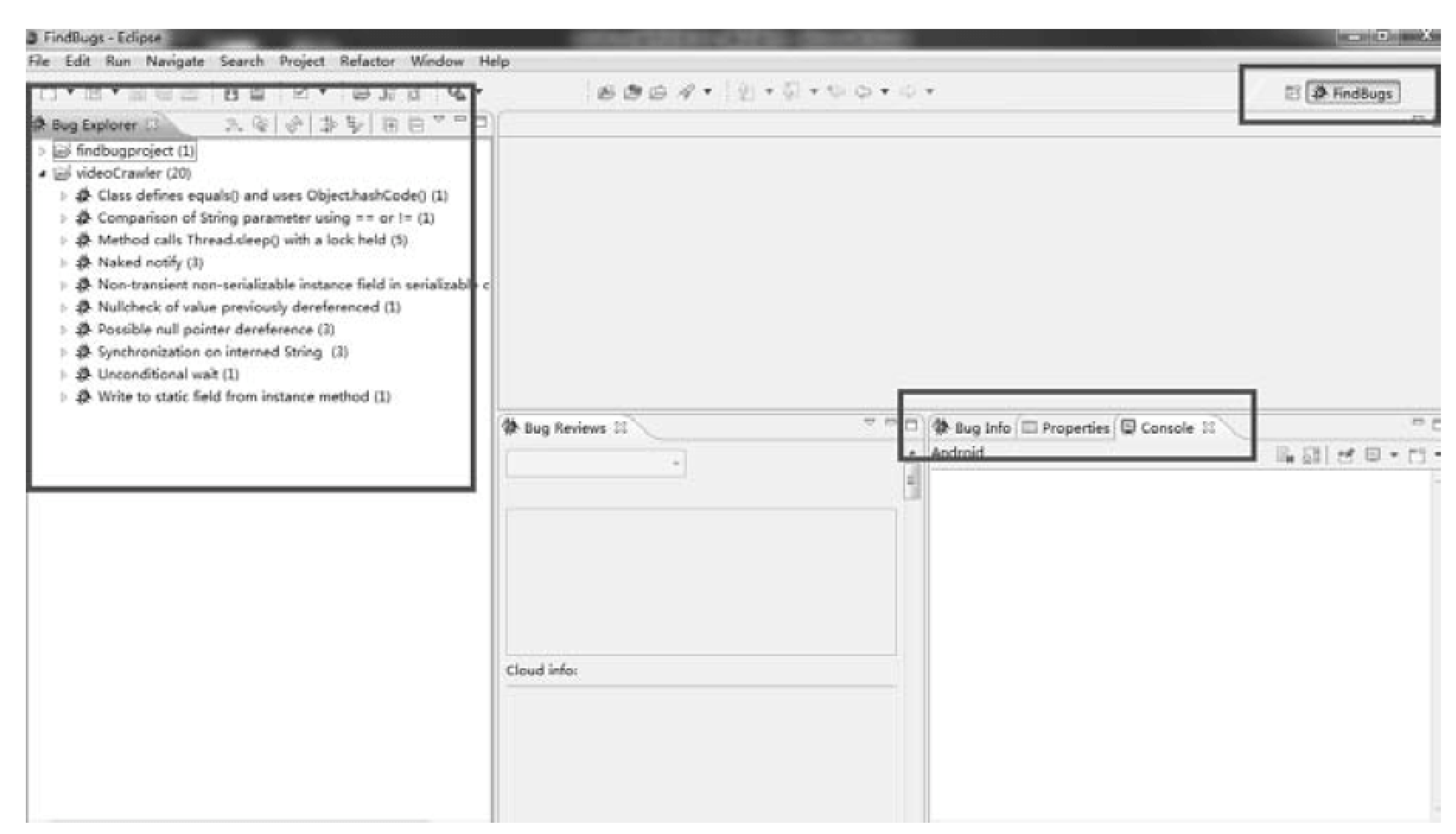


图 4-30　Bug 具体列表

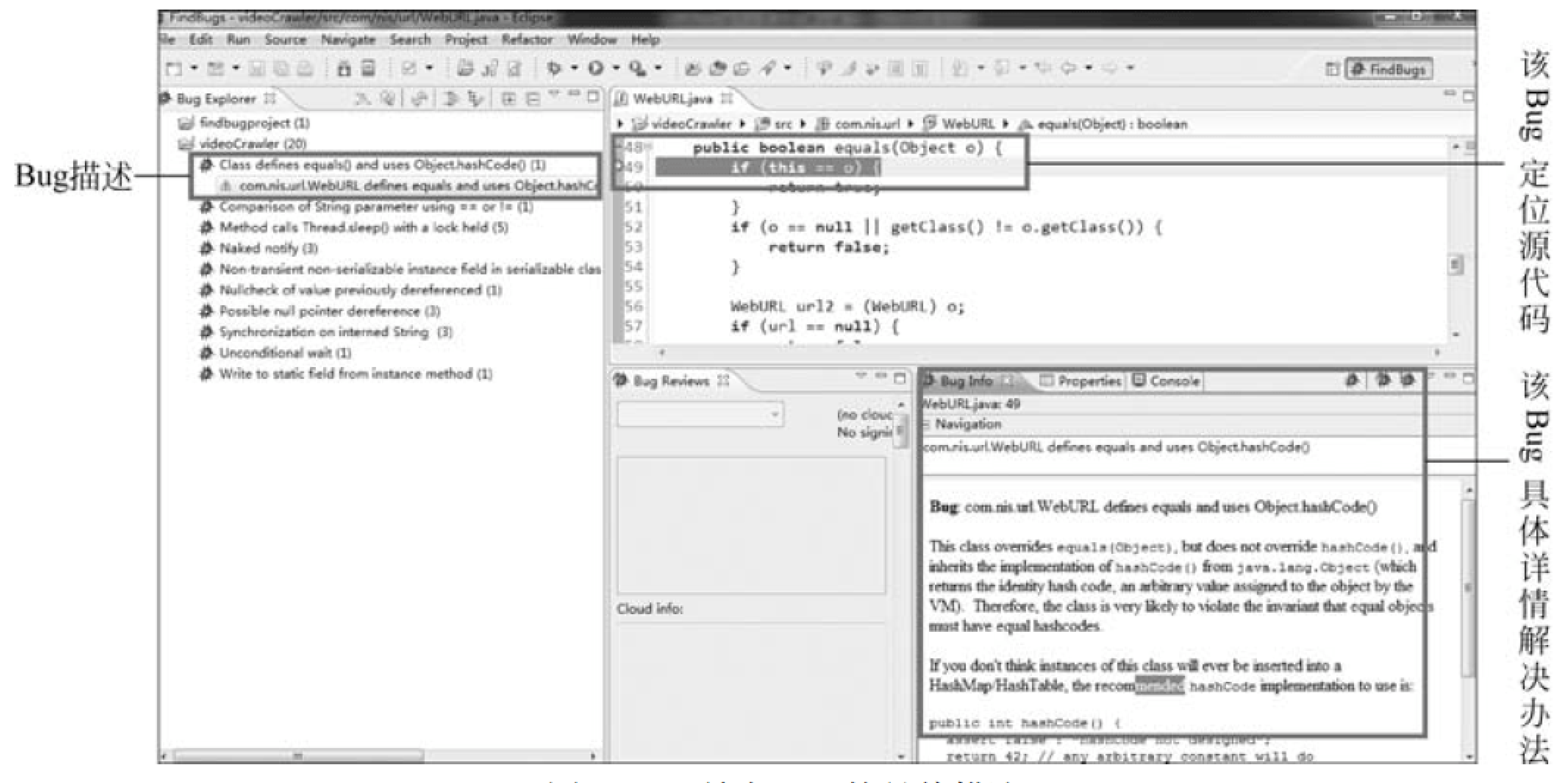


图 4-31　所选 Bug 的具体描述

4. 完成状态

项目根据 Bug 提示依次解决，Bug 标记将会消失。对于不需要解决的 Bug，可以右击项目选择 Find Bugs|Clear Bug Markers 命令进行消除。消除后，打开程序应用界面如图 4-32 所示。

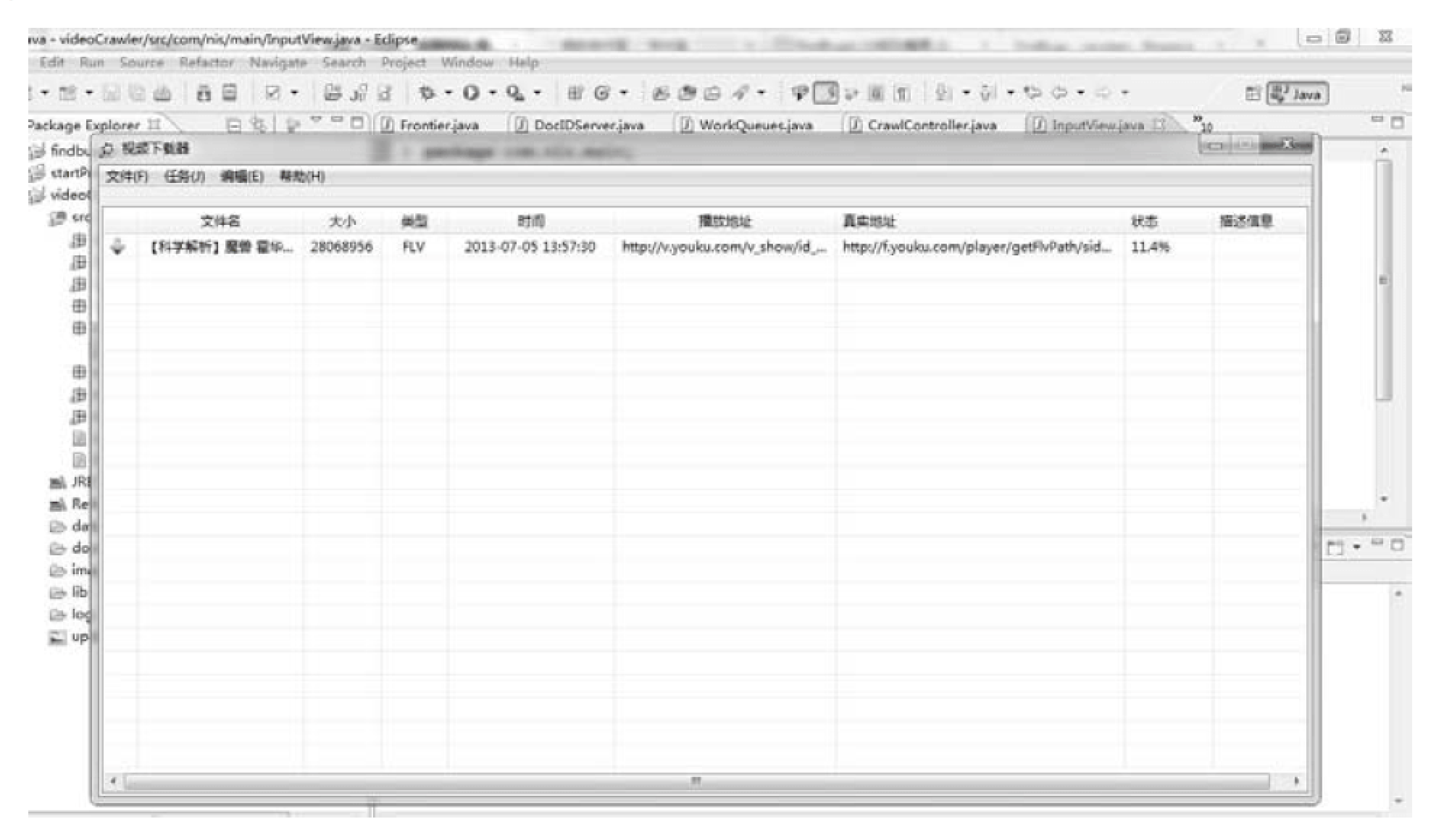


图 4-32　测试修改后程序界面

4.2.3　FindBugs 的 Bug 级别介绍

1. Bad practice 坏的实践

下面列举几个不好的实践。

(1) HE：类定义了 equals()，却没有 hashCode()；或类定义了 equals()，却使用 Object.hashCode()；或类定义了 hashCode()，却没有 equals()；或类定义了 hashCode()，却使用 Object.equals()；类继承了 equals()，却使用 Object.hashCode()。

(2) SQL：Statement 的 execute 方法调用了非常量的字符串；或 Prepared Statement 是由一个非常量的字符串产生。

(3) DE：方法终止或不处理异常，一般情况下，异常应该被处理或报告，或被方法抛出。

2. Correctness 一般的正确性问题

下面列举几个可能导致错误的代码。

(1) NP：空指针被引用；在方法的异常路径里，空指针被引用；方法没有检查参数是否为 null；null 值产生并被引用；null 值产生并在方法的异常路径被引用；传给方法一个声明为@NonNull 的 null 参数；方法的返回值声明为@NonNull 但实际是 null。

(2) Nm：类定义了 hashcode()方法，但实际上并未覆盖父类 Object 的 hashCode()；类定义了 tostring()方法，但实际上并未覆盖父类 Object 的 toString()；很明显的方法和构造器

混淆；方法名容易混淆。

(3) SQL：方法尝试访问一个 Prepared Statement 的 0 索引；方法尝试访问一个 ResultSet 的 0 索引。

(4) UwF：所有的 write 都把属性置成 null，这样所有的读取都是 null，这样这个属性是否有必要存在；或属性从没有被 write。

3. Internationalization 国际化

当对字符串使用 upper 或 lowercase 方法时，如果是国际的字符串，可能会不恰当地转换。

4. Malicious code vulnerability 可能受到的恶意攻击

如果代码公开，可能受到恶意攻击的代码，下面列举几个。

(1) FI：一个类的 finalize()应该是 protected，而不是 public 的。

(2) MS：属性是可变的数组；属性是可变的 Hashtable；属性应该是 package protected 的。

5. Multithreaded correctness 多线程的正确性

下面列举几个多线程编程时，可能导致错误的代码。

(1) ESync：空的同步块，很难被正确使用。

(2) MWN：错误使用 notify()，可能导致 IllegalMonitorStateException 异常；或错误地使用 wait()。

(3) No：使用 notify()而不是 notifyAll()，只是唤醒一个线程而不是所有等待的线程。

(4) SC：构造器调用了 Thread.start()，当该类被继承可能会导致错误。

6. Performance 性能问题

下面列举几个可能导致性能不佳的代码。

(1) DM：方法调用了低效的 Boolean 的构造器，而应该用 Boolean.valueOf(…)；用类似 Integer.toString(1) 代替 new Integer(1).toString()；方法调用了低效的 float 的构造器，应该用静态的 valueOf 方法。

(2) SIC：如果一个内部类想在更广泛的地方被引用，它应该声明为 static。

(3) SS：如果一个实例属性不被读取，考虑声明为 static。

(4) UrF：如果一个属性从没有被 read，考虑从类中去掉。

(5) UuF：如果一个属性从没有被使用，考虑从类中去掉。

7. Dodgy 危险的

下面列举几个具有潜在危险的代码，可能在运行期产生错误。

(1) CI：类声明为 final 但声明了 protected 的属性。

(2) DLS：对一个本地变量赋值，但却没有读取该本地变量；本地变量赋值成 null，却没有读取该本地变量。

(3) ICAST：整型数字相乘结果转化为长整型数字，应该将整型先转化为长整型数字再相乘。

(4) INT：没必要的整型数字比较，如 X <= Integer.MAX_VALUE。

(5) NP：对 readline()的直接引用，而没有判断是否 null；对方法调用的直接引用，而方法可能返回 null。

(6) REC：直接捕获 Exception，而实际上可能是 RuntimeException。

(7) ST：从实例方法里直接修改类变量，即 static 属性。

4.3　代码静态分析工具 PMD

PMD 是由 DARPA 在 SourceForge 上发布的开源 Java 代码静态分析工具。最初，PMD 是为了支持 Cougaar 项目而开发的。Cougaar 是美国国防高级研究计划局（Defense Advanced Research Projects Agency，DARPA）的一个项目。PMD 通过其内置的编码规则对 Java 代码进行静态检查，主要包括对潜在的 Bug、未使用的代码、重复的代码、循环体创建新对象等问题的检验。PMD 提供了和多种 Java IDE 的集成，例如 Eclipse、IDEA、NetBean 等。

PMD 是一种开源分析 Java 代码错误的工具。与其他分析工具不同的是，PMD 通过静态分析获知代码错误。也就是说，在不运行 Java 程序的情况下报告错误。PMD 附带了许多可以直接使用的规则，利用这些规则可以找出 Java 源程序的许多问题，例如以下这些错误。

(1) 潜在的 Bug：空的 try/catch/finally/switch 语句。

(2) 未使用的代码：未使用的局部变量、参数、私有方法等。

(3) 可选的代码：String/StringBuffer 的滥用。

(4) 复杂的表达式：不必需的 if 语句、可以使用 while 循环完成的 for 循环。

(5) 重复的代码：复制/粘贴代码意味着复制/粘贴 Bug。

(6) 循环体创建新对象：尽量不要复制 for 或 while 循环体内实例化一个新对象。

(7) 资源关闭：Connect，Result，Statement 等使用之后确保关闭掉。

此外，用户还可以自己定义规则，检查 Java 代码是否符合某些特定的编码规范。例如，可以编写一个规则，要求 PMD 找出所有创建 Thread 和 Socket 对象的操作。

4.3.1　PMD 功能介绍

1. 实现原理

PMD 的核心是 JavaCC 解析器生成器。PMD 结合运用 JavaCC 和 EBNF（Extended Backus-Naur Formal，扩展巴科斯-诺尔范式）文法产生一个分析器，用来分析 Java 源代码（文本）。又在 JavaCC 的基础上加入了语义的概念也就是 JJTree，这样就把 Java source 转换成了一个抽象语法树（Abstract Syntax Tree，AST），AST 是一个结构化的对象层次结构。可以用访问者模式访问这个结构上的每个节点。从而找出哪个节点违反了哪些规则。

2. 实现过程

首先传一个文件名或者 Ruleset 给 PMD，PMD 把该文件流传给自己生成的 JavaCC 分析器分析完毕后，PMD 获得了分析生成的 AST 的一个引用。PMD 把 AST 处理成一个符号表，用户可以在符号表里面查询一些有用的信息，每个 PMD 规则都会遍历整个 AST 并检验是否发生了错误，接着 PMD 产生一个报表，上面说明了有哪些地方违反了 PMD 规则。

3. PMD 规则

1) PMD 默认规则

PMD 自带了很多规则集合，并且分类写入不同的 ruleset 文件。

2) PMD 自定义规则

PMD 自带了很多代码规范的规则，还可以自定义规则，可以把这些规则整合到一起，最后，运行 PMD 的时候就可以指定这个 ruleset 文件，按照需求进行代码检查。

4. 支持的 Java 编辑器

PMD 支持的编辑器包括：JDeveloper、Eclipse、JEdit、JBuilder、BlueJ、CodeGuide、NetBeans/Sun Java Studio Enterprise/Creator、Intelli J IDEA、TextPad、Maven、Ant、Gel、JCreator 和 Emacs。不同编辑器对应 PMD 插件版本可在 http://sourceforge.net/projects/pmd/files/进行获取。

5. 发版计划

PMD 计划每一两个月发布一个正式版本。2013 年 5 月 1 日发布了最新的 PMD 5.0.4 版，同时在 2013 年 5 月 10 日发布了 PMD for Eclipse 4.0.0.v20130510-1000 版。

6. PMD 源码获取

(1) 通过官网获取源代码：http://sourceforge.net/projects/pmd。

(2) 通过 Git Hub 获取源代码：https://github.com/pmd。

4.3.2　PMD 环境建立

可以从 PMD 的网站下载 PMD 的二进制版本，或下载带源代码的版本，下载得到的都是 ZIP 文件。PMD 源码可通过下载地址http://sourceforge.net/projects/pmd/files/pmd/获取。

如果要在一个 Java 源代码目录中运行 PMD，只需直接在命令行上运行下面的命令：

```
G:\pmd-bin-5.0.4\bin>java –jar ..\lib\pmd-5.0.4.jar G:\cellstyle text rulesets/unusedcode.xml
```

一些可选参数如下。

-debug：打印 debug 日志信息。

-targetjdk：指定目标源代码的版本——1.3, 1.4, 1.5, 1.6 或 1.7；默认是 1.5。

-cpus：指定创建的线程数。

-encoding：指定 PMD 检查的代码的编码方式。

-excludemarker：指定 PMD 需要忽略的行的标记，默认为 NOPMD。

-shortnames：在报告中显示缩短的文件名。

-linkprefix：HTML 源文件的路径，只是为了 HTML 显示。

-lineprefix：自定义的锚，用于影响源文件中的行，只是用于 HTML 显示。

-minimumpriority：规则的优先级限制，低于优先级的规则将不被使用。

-nojava：不检查 Java 文件，默认是检查 Java 文件。

-jsp：检查 JSP/JSF 文件，默认不检查。

-reportfile：将报告输出到文件，默认是打印在控制台。

-benchmark：输出一个基准清单，默认输出到控制台。

-xslt：覆盖默认的 xslt。

-auxclasspath：指定源代码文件使用的类路径。

在 Eclipse 中安装 PMD 插件的方法有如下两种。

1. 通过 Eclipse 更新安装

PMD 可作为插件集成到很多流行的 IDE 中，很多的插件中都包含 PMD 的 jar 文件，这个 jar 文件中包含规则集。所以虽然一些插件中使用 rulesets/unusedcode.xml 作为参数引用规则集，但是实际上是使用 getResourceAsStream()方法来从 PMD 的 jar 文件中加载。

由于 Eclipse 是比较流行的开源 Java/J2EE 开发 IDE，所以本文主要介绍如何在 Eclipse 中使用 PMD 工具进行代码的检查,并且选用最新 JDK 1.7+Eclipse 4.2。在 Eclipse 中安装 PMD 的过程如图 4-33 所示。

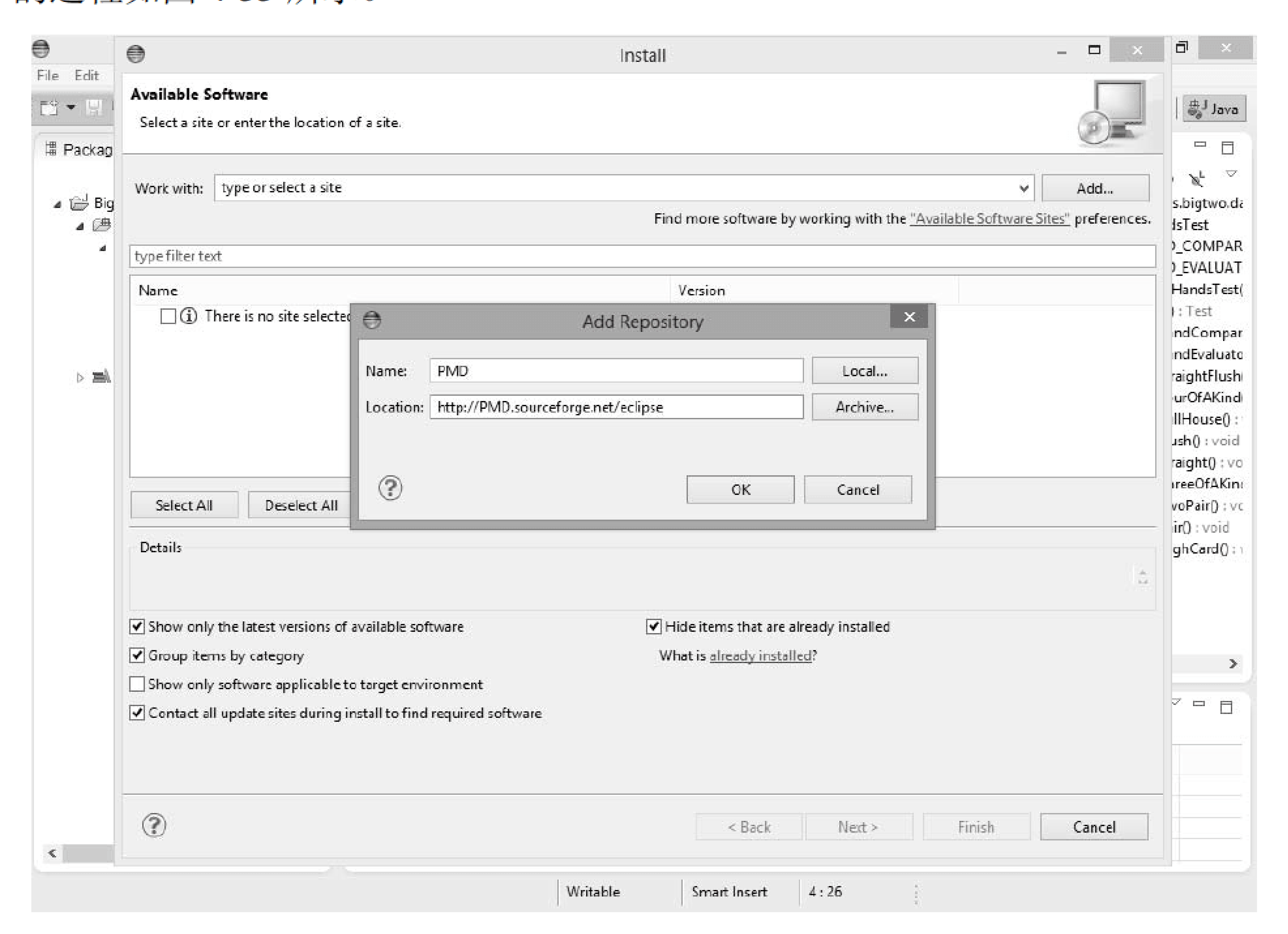


图 4-33　在 Eclipse 中安装 PMD



图 4-33　(续)

最后重启 Eclipse，PMD 即安装成功。

2. 本地安装

下载最新的 zip 文件包，然后执行上述过程，只是在 Add Repository 时，选择 Archive 来代替 Update Site，并使用下载的 zip 文件。

在安装完更新后，如果发生了一个异常，例如“java.lang.RuntimeException：Could not find that class xxxx”，这时可试着删除 workspace 中的.metadata/plugins/net.sourceforge.pmd.eclipse 目录下的 ruleset.xml 文件。

以下为主要的 Ant 配置信息：

```
<path id="pmd.path">
    <filesetdir="${lib.dir}/pmd-5.0.4">
        <include name="**/*.jar" />
    </fileset>
</path>
<taskdefname="pmd" classname="net.sourceforge.pmd.ant.PMDTask"
    classpathref="pmd.path" />
<taskdefname="cpd" classname="net.sourceforge.pmd.cpd.CPDTask"
    classpathref="pmd.path" />
<target name="pmd">
    <pmdshortFilenames="true">
        <ruleset>rulesets/favorites.xml</ruleset>
        <formatter type="html" toFile="d:\foo.html" toConsole="false" />
        <filesetdir="${src.dir}">
            <include name="**/*.java" />
        </fileset>
    </pmd>
</target>
<target name="cpd">
    <cpdminimumTokenCount="100" outputFile="d:/cpd.txt">
        <filesetdir="${src.dir}">
            <includename="**/*.java"/>
        </fileset>
    </cpd>
</target>
```

用 Ant 命令运行 build.xml，PMD 就会按照设定好的规则自动执行代码检查。

4.3.3　PMD 应用流程

1. PMD 应用流程图

PMD 应用流程图如图 4-34 所示。

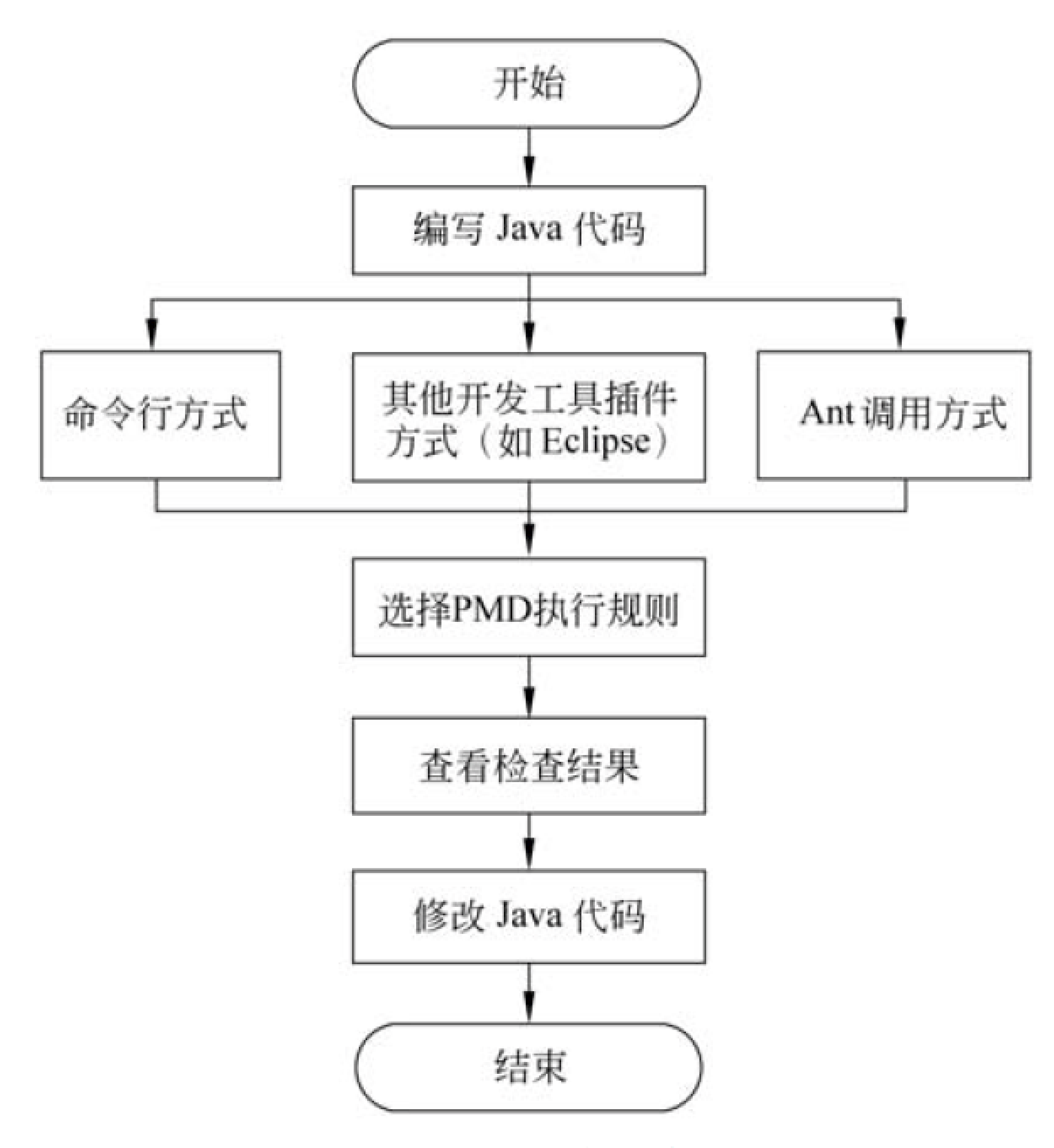


图 4-34　PMD 应用流程图

2. PMD 的使用流程

PMD 的使用流程概括起来可以分为以下几步。

1) PMD 的执行参数设置

启动 Eclipse IDE，打开工程，选择 Windows|Preferences 下的 PMD 项，可以对 PMD 的一些执行参数进行设置，如图 4-35 所示。

2) 配置 PMD 的检查规则

图 4-34 中 Rules Configuration 项目可以配置 PMD 的检查规则，自定义检查规则也可以在此通过 Import 的方式导入到 PMD 中，如图 4-36 所示。

3) 检查 Java 程序代码

配置好后，右击工程中需要检查的 Java Source，选择 PMD|Check Code With PMD，之后 PMD 就会通过规则检查 JavaSource，并将信息显示在 PMD 自己的视图上，如图 4-37 所示。

以上代码 PMD 会检查出：代码中出现 System.out.print 等警告描述。

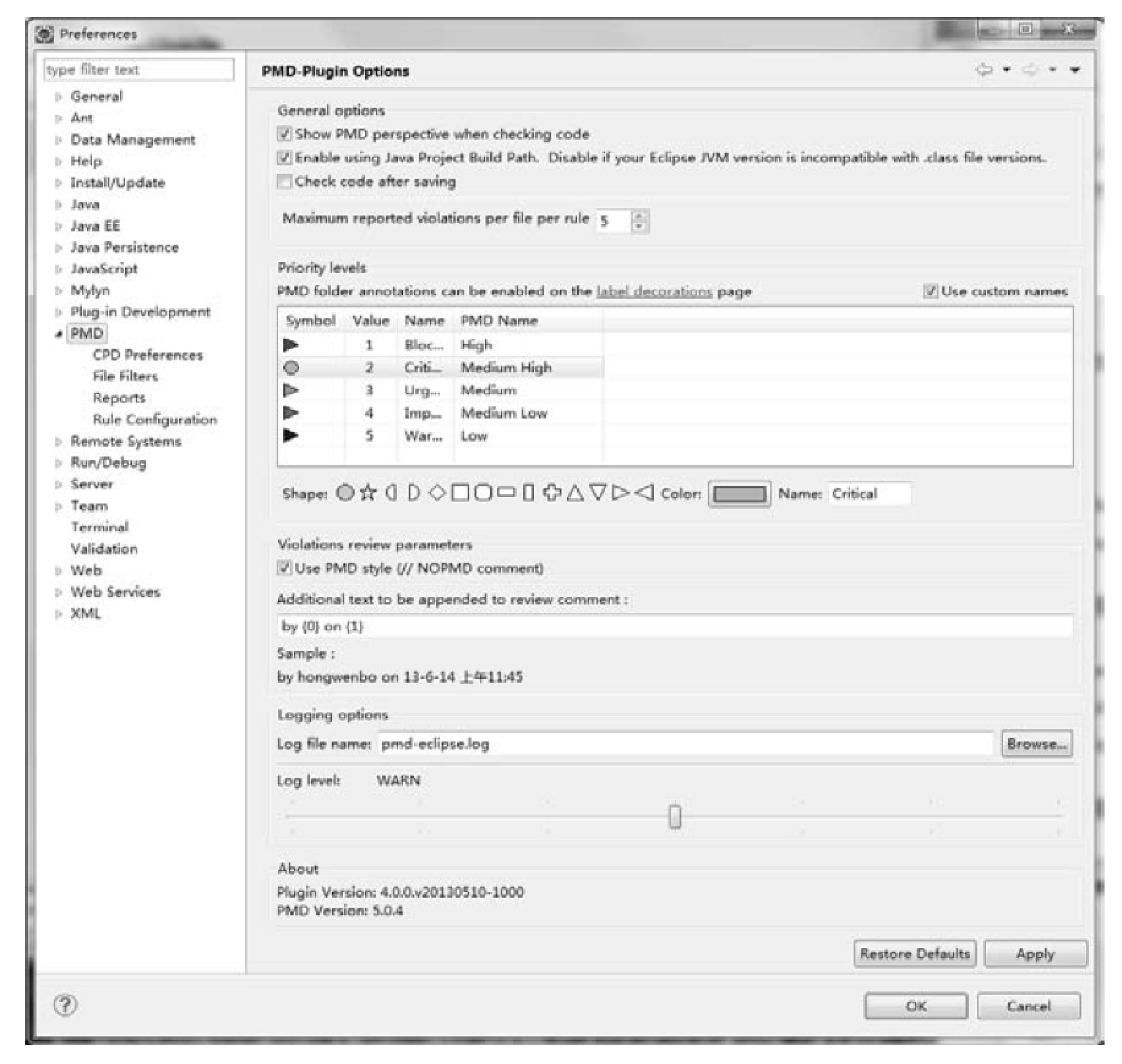


图 4-35　设置 PMD 执行参数

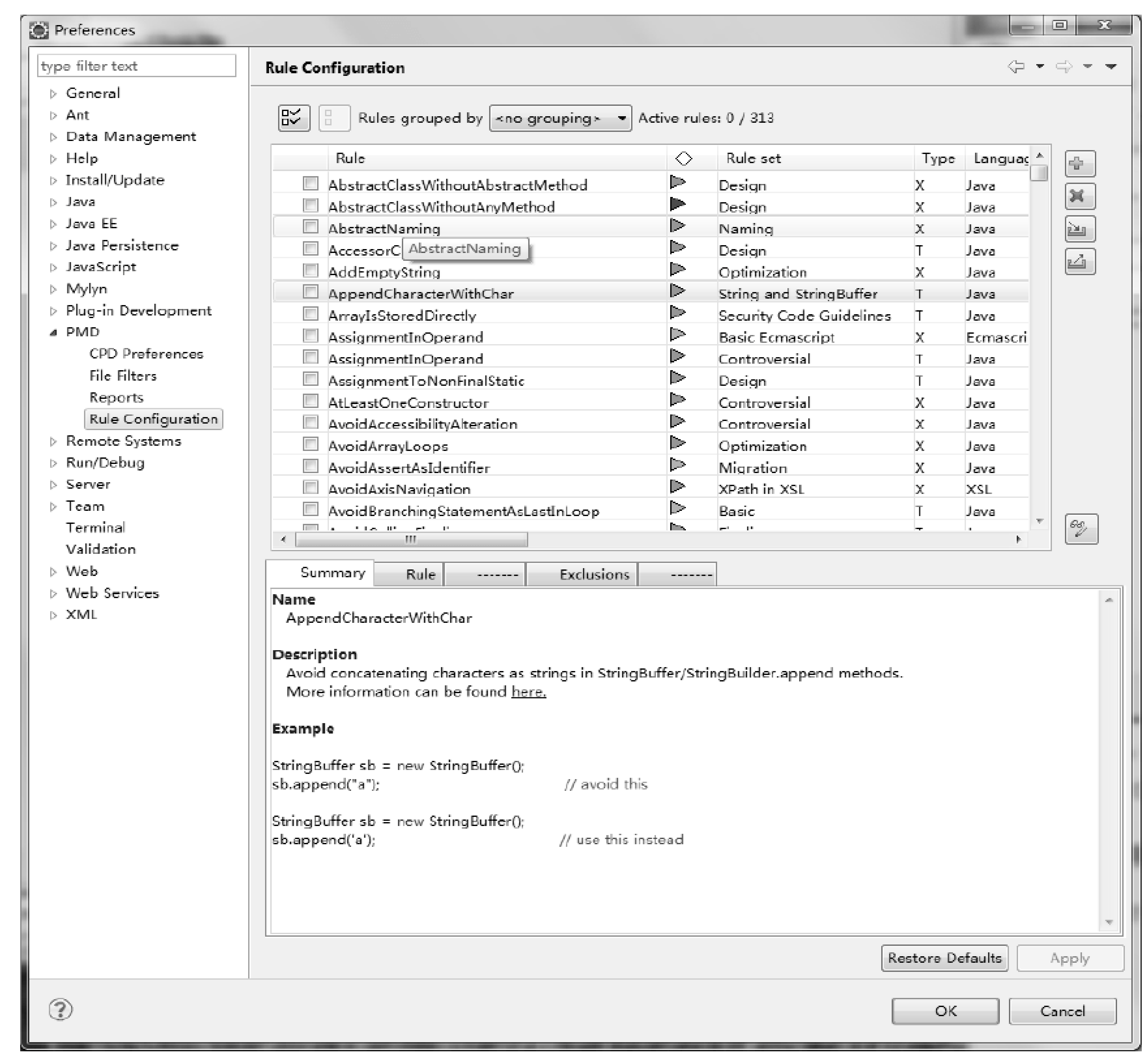


图 4-36　检查规则配置

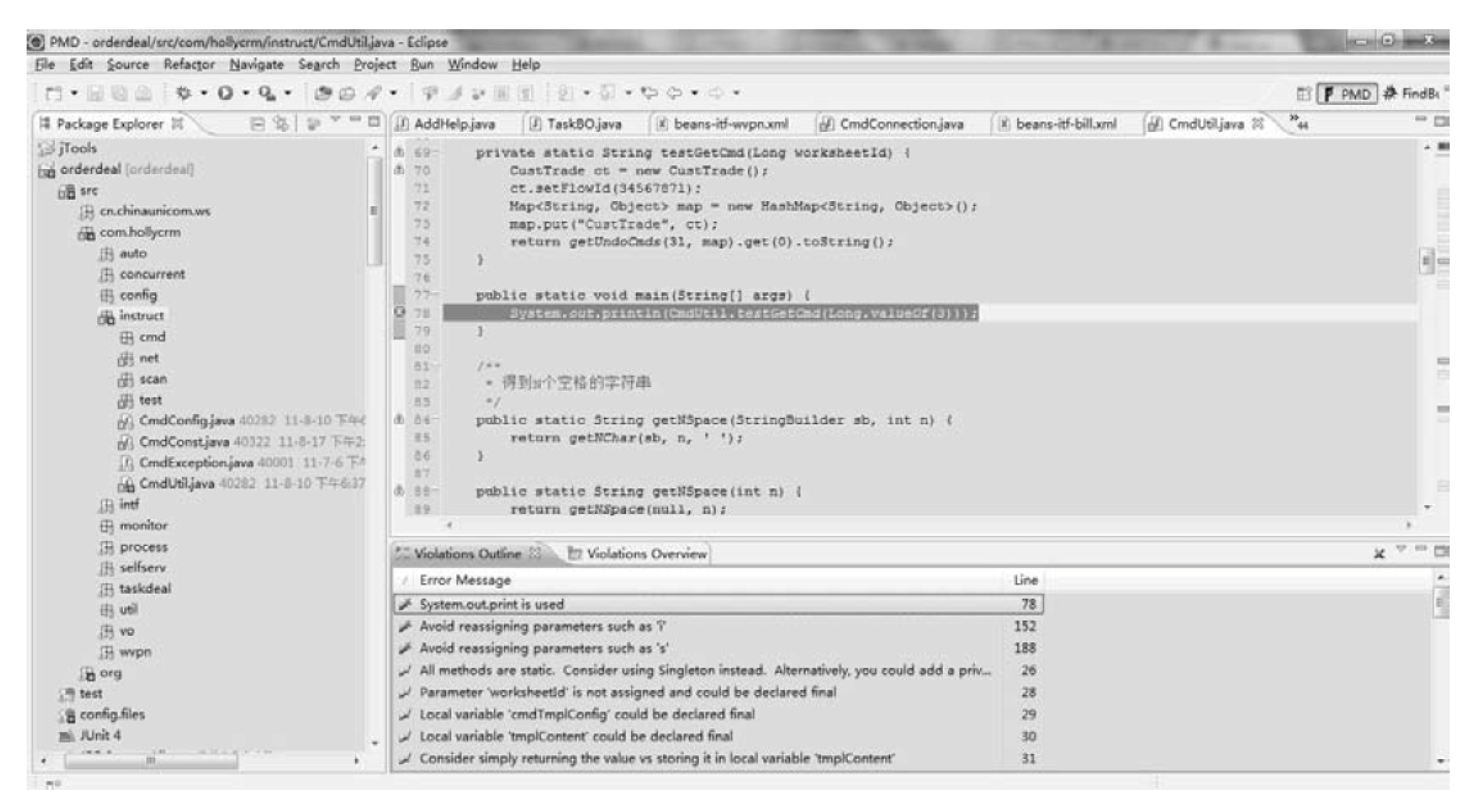


图 4-37　Java 源代码检查

4) 检查代码重复

PMD 的 CPD 能够帮助发现代码的重复，如图 4-38 所示。

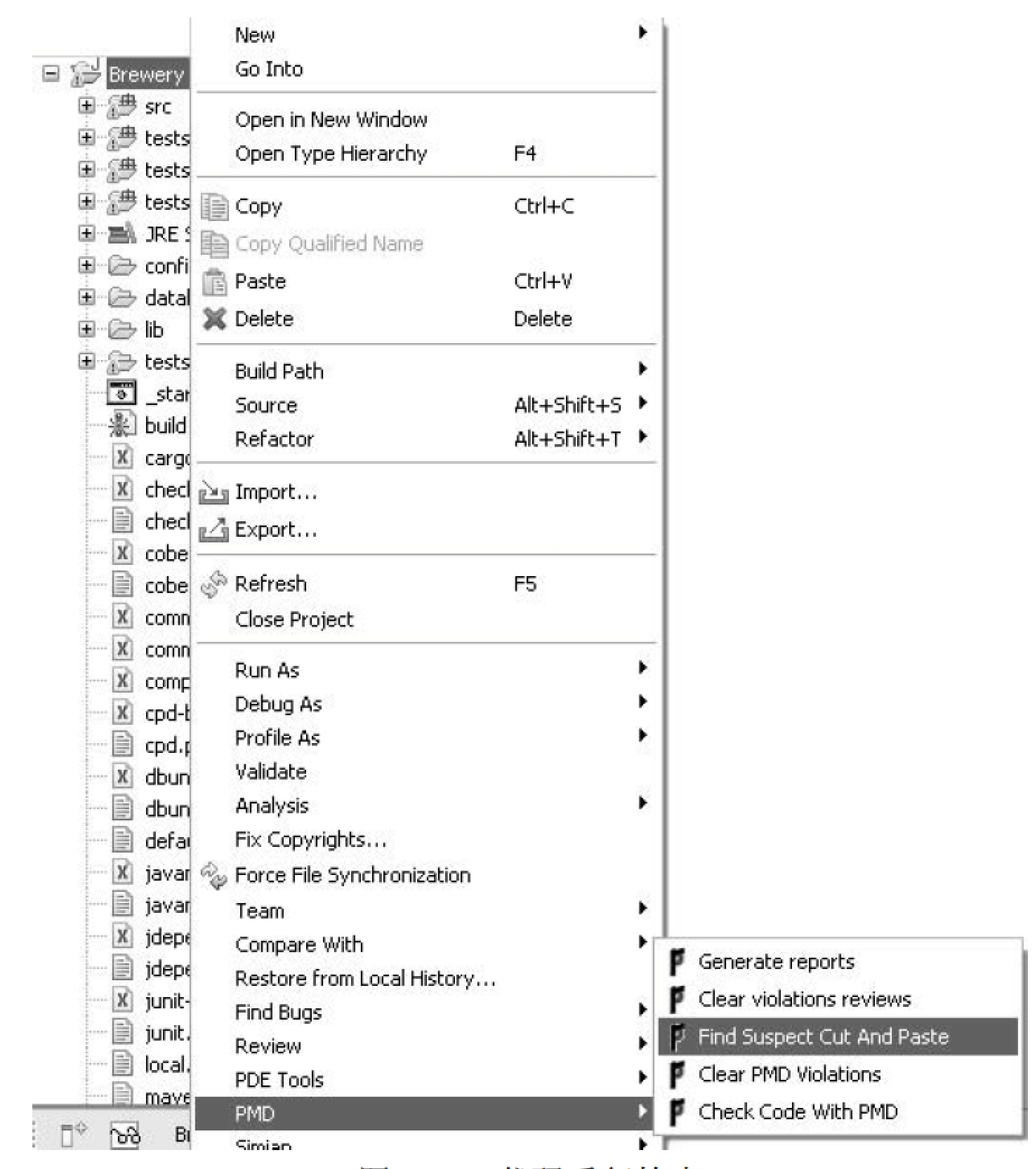


图 4-38　代码重复检查

一旦运行了 CPD，Eclipse 根目录下就会创建出一个 report 文件夹，其中包含一个叫

作 cpd.txt 的文件，文件中列示了所有重复的代码，如图 4-39 所示。

```
d:\workspace\orderdeal\reports\cpd-report.txt - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(T) 宏(O) 运行(R) 插件(P) 窗口(W) ?
cpd-report.txt | hosts | test.py | LocalSettings.php | FCKeditorSajax.body.php | ProfilerStub.php

Found a 237 line (735 tokens) duplication in the following files:
Starting at line 261 of D:\workspace\orderdeal\src\com\hollycrm\vo\CustOrderChange.ja
Starting at line 293 of D:\workspace\orderdeal\src\com\hollycrm\vo\CustOrder.java

    public void setProcInsId(String procInsId) {
        this.procInsId = procInsId;
    }

    public Date getReqFinish() {
        return this.reqFinish;
    }

    public void setReqFinish(Date reqFinish) {
        this.reqFinish = reqFinish;
    }

    public Date getCompletedTime() {
        return this.completedTime;
    }
```

图 4-39　重复代码显示

3. 应用举例

可以用 PMD 对下面的源码进行分析，该源码中存在一些缺陷，如图 4-40 所示。

```
package pmd.test;

import java.io.*;

public class Test {

    public boolean copy(InputStream is, OutputStream os) throws IOException {
        int count = 0;
        byte[] buffer = new byte[1024];
        while ((count = is.read(buffer)) >= 0) {
            os.write(buffer, 0, count);
        }
        return true;
    }
    public void copy(String[] a, String[] b, String ending) {
        int index;
        String temp = null;
        int length = temp.length();
        for (index = 0; index <a.length; index++) {
            if (true) {
                if (temp == ending) {
                    break;
                }
                b[index] = temp;
```

```
            }
        }
    }
    public void readFile(File file) {
        InputStream is = null;
        OutputStream os = null;
        try {
            is = new BufferedInputStream(new FileInputStream(file));
            os = new ByteArrayOutputStream();
            copy(is, os);
            is.close();
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
        }
    }
}
```

```
14 public class Test {
15 
16     public boolean copy(InputStream is, OutputStream os) throws IOException {
17         int count = 0;
18         byte[] buffer = new byte[1024];
19         while ((count = is.read(buffer)) >= 0) {
20             os.write(buffer, 0, count);
21         }
22         return true;
23     }
24 
25     public void copy(String[] a, String[] b, String ending) {
26         int index;
27         String temp = null;
28         int length = temp.length();// 未使用变量和空指针错误
29         for (index = 0; index <a.length; index++) {
30             if (true) { // 多余的if语句
31                 if (temp == ending) { // 对象比较应使用equals
32                     break;
33                 }
34                 b[index] = temp; // 缺少数组下标越界检查
35             }
36         }
37     }
38     public void readFile(File file) {
39         InputStream is = null;
40         OutputStream os = null;
41         try {
42             is = new BufferedInputStream(new FileInputStream(file));
43             os = new ByteArrayOutputStream();
44             copy(is, os); // 未使用方法返回值
45             is.close();
46             os.close();
47         } catch (IOException e) {
48             e.printStackTrace(); // 可能造成I/O流未关闭
49         } finally {
50             // 空的try/catch/finally块
51         }
52     }
53 }
```

图 4-40　源代码中存在的问题

图 4-40 中显示源程序中有 7 个问题。

问题 1：第 28 行，应该提示空指标错误。

问题 2：第 28 行，变量 length 未使用，应该提示未使用变量。

问题 3：第 30 行，由于 if 判断条件 true，所以这里应该要提示多余的 if 语句。

问题 4：第 31 行，应该提示对象比较应使用 equals。

问题 5：第 34 行，应该提示缺少数组下标越界检查。

问题 6：第 48 行，应该避免直接使用 printStackTrace()，可能造成 I/O 流未关闭。

问题 7：第 50 行，应该提示空的 finally 代码块。

将上述源代码保存为 Test.java 文件，通过 PMD|Check Code 对其源码进行检测，检查结果如图 4-41 所示。

Violations Outline

Priority	Line	created	Rule	Error Message
	19	Fri Jun 14 23:53:20 CST 2013	AssignmentInOperand	Avoid assignments in operands
	29	Fri Jun 14 23:53:20 CST 2013	UnusedLocalVariable	Avoid unused local variables such as 'length'.
	29	Fri Jun 14 23:53:20 CST 2013	LawOfDemeter	Potential violation of Law of Demeter (object not created locally)
	32	Fri Jun 14 23:53:20 CST 2013	UnconditionalIfStatement	Do not use if statements that are always true or always false
	34	Fri Jun 14 23:53:20 CST 2013	CompareObjectsWithEquals	Use equals() to compare object references.
	54	Fri Jun 14 23:53:20 CST 2013	AvoidPrintStackTrace	Avoid printStackTrace(); use a logger call instead.
	55	Fri Jun 14 23:53:20 CST 2013	EmptyFinallyBlock	Avoid empty finally blocks
	55	Fri Jun 14 23:53:20 CST 2013	EmptyFinallyBlock	Avoid empty finally blocks

图 4-41　PMD 检查出的问题

违反 PMD 检查规则的统计列表如图 4-42 所示。

Violations Overview

Element	# Violations	# Violations/...	# Violations/...	Project
pmd.test	8	285.7	2.67	pmd.test
Test.java	8	285.7	2.67	pmd.test
AssignmentInOperand	1	35.7	0.33	pmd.test
UnconditionalIfStatement	1	35.7	0.33	pmd.test
UnusedLocalVariable	1	35.7	0.33	pmd.test
AvoidPrintStackTrace	1	35.7	0.33	pmd.test
CompareObjectsWithEquals	1	35.7	0.33	pmd.test
LawOfDemeter	1	35.7	0.33	pmd.test
EmptyFinallyBlock	2	71.4	0.67	pmd.test

图 4-42　PMD 规则的违反情况

从执行结果来看，问题 1、2、3、4、6、7 均被有效提示，只有问题 5：缺少数组下标越界检查未被检查出来，这是由于 PMD 只是通过静态分析来获知代码错误。也就是说，在不运行 Java 程序的情况下报告错误。而在 Java 语言中，数组下标越界属于运行时错误，故无法通过 PMD 报告错误。

4.4　开源代码静态分析工具 Splint

Splint 是一个动态检查 C 语言程序安全弱点和编写错误的程序。Splint 会进行多种常规检查，包括未使用的变量、类型不一致、使用未定义变量、无法执行的代码、忽略返回值、

执行路径未返回、无限循环等错误。

4.4.1 Splint 的安装

1. 在 Linux 下安装

1) rpm 安装

GTES 10.5 和 11 版本已经整合了 Splint 软件包，可以直接使用。

2) 源代码安装

下载地址：http://www.splint.org/downloads/splint-3.1.2.src.tgz。

源码包安装：

```
# tar zxvf splint-3.1.2.src.tgz
# cd splint-3.1.2
# ./configure
# make
# make install
# vi ~/.bashrc
```

添加：

```
export LARCH_PATH=/usr/local/splint/share/splint/lib
export LCLIMPORTDIR=/usr/local/splint/share/splint/import
# source ~/.bashrc
# export PATH=/usr/local/splint/bin/splint:$PATH
```

2. 在 Windows 下安装

(1) 解压。如果解压到 C:\splint-3.1.2，则不用调整环境变量，可执行文件在 bin 目录下。如果解压到了其他路径下，则需要修改环境变量。在用户变量中加上：

```
LARCH_PATH - <你安装 Splint 的路径>\lib
LCLIMPORTDIR - <你安装 Splint 的路径>\imports
include - 系统 include 文件所在的目录，如 C:\Dev-Cpp\include
```

如果 Lint 时显示找不到 stdio.h 等，就是这里没有设定好。如果不写 include，则需要在 Splint 的参数中加上-I"C:\Dev-Cpp\include"。

(2) 在用户变量 Path 中加上 splint.exe 所在的路径，以方便以后调用。

(3) 使以上修改生效：注销当前用户后重新登录。

4.4.2　Splint 的应用

1. 空引用错误

在引用没有指向任何内存地址的指针时，会导致使用了一个没有赋值的指针的错误。Splint 支持一种特别的注释，这种注释写在 C 程序代码中，用于对程序进行特殊说明。例如下面这段程序使用/*@null@*/进行了说明，表明*s 的值可能会是 null。

示例 4-1：

```
//null.c
char firstChar1 (/*@null@*/ char *s)
{
return *s;
}
char firstChar2 (/*@null@*/ char *s)
{
 if (s ==NULL) return '\0';
 return *s;
}
//END
```

使用 Splint 扫描这个程序时，会输出:

```
# splint null.c
Splint 3.1.1 --- 28 Apr 2005
null.c: (in function firstChar1)
null.c:3:11: Dereference of possibly null pointer s: *s
null.c:1:35: Storage s may become null
Finished checking --- 1 code warning found
```

由于 firstChar1 和 firstChar2 都使用了 null 说明，以表示指针 s 可能是个 null 值；所以，Splint 会对 s 值的使用情况进行检查。因为 firstChar2 函数中对 s 的值进行了 null 的判断，所以没有对 firstChar2 函数的指针 s 输出警告信息。

2. 未定义的变量错误

在 C 语言中，要求先定义变量，而后才可以使用。所以，当使用一个没有定义的变量时，编译就会出错。如下例所示，使用/*@in@*/说明的变量，表示必须进行定义。使用/*@out@*/说明的变量，表示在执行过此函数后，这个变量就进行了定义。

示例 4-2：

```
// usedef.c
```

```
extern void setVal (/*@out@*/ int *x);
extern int getVal (/*@in@*/ int *x);
extern int mysteryVal (int *x);
int dumbfunc (/*@out@*/ int *x, int i)
 {
 if (i > 3) return *x;
 else if (i > 1)
    return getVal (x);
 else if (i == 0)
    return mysteryVal (x);
 else
  {
 setVal (x);
 return *x;
  }
 }
// END
```

使用 Splint 检查 usedef.c：

```
$ splint usedef.c
Splint 3.1.1 --- 28 Apr 2005
usedef.c: (in function dumbfunc)
usedef.c:7:19: Value *x used before definition
   An rvalue is used that may not be initialized to a value on some execution
   path. (Use -usedef to inhibit warning)
usedef.c:9:18: Passed storage x not completely defined (*x is undefined):
                          getVal (x)
   Storage derivable from a parameter,  return value or global is not defined.
   Use /*@out@*/ to denote passed or returned storage which need not be defined.
   (Use -compdef to inhibit warning)
usedef.c:11:22: Passed storage x not completely defined (*x is undefined):
                           mysteryVal (x)
Finished checking --- 3 code warnings
```

错误原因：由于程序中没有对 x 进行定义，所以报未定义错误。但 setVal()使用了/*@out@*/说明，所以在语句“setVal(x);”和“return x;”中，没有报未定义错误。

3. 类型错误

C 语言中的数据类型较多，各自之间有些细微差别。Splint 还可以对变量类型进行检查。示例 4-3：

```
//bool.c
```

```
int f (int i,  char *s, bool b1,  bool b2)
{
if (i = 3) return b1;
if (!i || s) return i;
if (s) return 7;
if (b1 == b2)
return 3;
return 2;
}
//END
```

使用 Splint 进行检查:

```
$ splint bool.c
Splint 3.1.1 --- 28 Apr 2005
bool.c: (in function f)
bool.c:4:5: Test expression for if is assignment expression: i = 3
  The condition test is an assignment expression. Probably,  you mean to use ==
  instead of =. If an assignment is intended,  add an extra parentheses nesting
  (e.g.,  if ((a = b)) ...) to suppress this message. (Use -predassign to
  inhibit warning)
// 错误原因: if语句中的条件表达式是一个赋值语句。
bool.c:4:5: Test expression for if not boolean,  type int: i = 3
     Test expression type is not boolean or int. (Use -predboolint to inhibit warning)
// 错误原因: if语句中的条件表达式的返回值, 不是布尔型, 而是整型。
bool.c:4:8: Return value type bool does not match declared type int: b1
     Types are incompatible. (Use -type to inhibit warning)
// 错误原因: 返回值是布尔型, 而不是整型。
bool.c:5:6: Operand of ! is non-boolean (int): !i
     The operand of a boolean operator is not a boolean. Use +ptrnegate to allow !
     to be used on pointers. (Use -boolops to inhibit warning)
// 错误原因: "!"操作符的操作数不是布尔型, 而是整型 i。
bool.c:5:11: Right operand of || is non-boolean (char *): !i || s
// 错误原因: "||"操作符的右操作数不是布尔型, 而是字符指针。
bool.c:7:5: Use of == with boolean variables (risks inconsistency because of
                    multiple true values): b1 == b2
      Two bool values are compared directly using a C primitive. This may produce
      unexpected results since all non-zero values are considered true,  so
      different true values may not be equal. The file bool.h (included in
      splint/lib) provides bool_equal for safe bool comparisons. (Use -boolcompare
      to inhibit warning)
// 错误原因: 不应该使用"= ="对两个布尔型进行比较, 应该使用"&&"。
```

```
Finished checking --- 6 code warnings
```

示例 4-4：

```
//malloc1.c
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
char *some_mem;
int size1=1048576;
some_mem=(char *)malloc(size1);
printf("Malloed 1M Memory!\n");
free(some_mem);
exit(EXIT_SUCCESS);
}
//END
```

使用 Splint 检查 malloc1.c：

```
$ splint malloc1.c
Splint 3.1.1 --- 28 Apr 2005
malloc1.c: (in function main)
malloc1.c:9:25: Function malloc expects arg 1 to be size_t gets int: size1
   To allow arbitrary integral types to match any integral type，  use
   +matchanyintegral.
Finished checking --- 1 code warning
```

修改变量 size1 的定义为：

```
size_t size1=1048576;
```

再使用 Splint 进行检查：

```
$ splint malloc1.c
Splint 3.1.1 --- 28 Apr 2005
Finished checking --- no warnings
```

没有检查到错误。

4. 内存检查

缓冲区溢出错误是一种非常危险的 C 语言错误，大部分安全漏洞都与它有关。Splint 可以对缓冲区的使用进行检查，报告溢出或越界错误。

示例 4-5：

```
//over.c
int main()
{
int buf[10];
buf[10] = 3;
return 0;
}
//END
```

使用 Splint 进行检查：

```
$ splint over.c    +bounds +showconstraintlocation
Splint 3.1.1 --- 21 Apr 2006
Command Line: Setting +showconstraintlocation redundant with current value
over.c: (in function main)
over.c:6:3: Likely out-of-bounds store:
    buf[10]
    Unable to resolve constraint:
    requires 9 >= 10
     needed to satisfy precondition:
    requires maxSet(buf @ over.c:6:3) >= 10
  A memory write may write to an address beyond the allocated buffer. (Use
  -likely-boundswrite to inhibit warning)
Finished checking --- 1 code warning
```

数组 buf 的大小是 10B，最大可使用的元素位置为 buf[9]，但程序中使用了 buf[10]，所以报错。

示例 4-6：

```
// bound.c
void updateEnv(char *str)
{
  char *tmp;
  tmp = getenv("MYENV");
  if(tmp != NULL) strcpy(str, tmp);
}
void updateEnvSafe(char *str, size_t strSize)
{
  char *tmp;
  tmp = getenv("MYENV");
  if(tmp != NULL)
```

```
        {
            strncpy(str，  tmp，  strSize -1);
            str[strSize - 1] = '/0';
        }
}
//END
```

使用 Splint 进行检查：

```
$ splint bound.c +bounds +showconstraintlocation
Splint 3.1.1 --- 21 Apr 2006
Command Line: Setting +showconstraintlocation redundant with current value
bound.c: (in function updateEnv)
bound.c:6:19: Possible out-of-bounds store:
    strcpy(str, tmp)
    Unable to resolve constraint:
    requires maxSet(str @ bound.c:6:26) >= maxRead(getenv("MYENV") @
    bound.c:5:9)
     needed to satisfy precondition:
    requires maxSet(str @ bound.c:6:26) >= maxRead(tmp @ bound.c:6:30)
     derived from strcpy precondition: requires maxSet(<parameter 1>) >=
    maxRead(<parameter 2>)
    A memory write may write to an address beyond the allocated buffer. (Use
    -boundswrite to inhibit warning)
```

错误原因：由于使用 strcpy 函数时，没有指定复制字符串的长度，所以可能导致缓冲区溢出。后面的 updateEnvSafe 函数使用了 strncpy 进行字符串复制，避免了这种情况发生。

4.4.3 Splint 与 IDE 的集成

1. UltraEdit

选择“高级”|“工具栏配置”。

在菜单项目名称中输入“splint”，如果配置过 Path，在命令行上直接输入“splint %F”即可，如果没有则需给出完整路径。

2. Source Insight

选择 Options|Custom Commonds。

在 Commond 中输入“splint”，如果配置过 Path，在命令行上直接输入“splint %F”即可，如果没有则需给出完整路径。

3. Splint 的图形用户界面

Christoph Thielecke 在 2008 年年底为 Splint 开发了一个图形用户界面，下载地址：

http://crissi.linux-administrator.com/linux/splintgui/index_en.html。图 4-43 和图 4-44 给出了 Splint GUI 的使用情况。

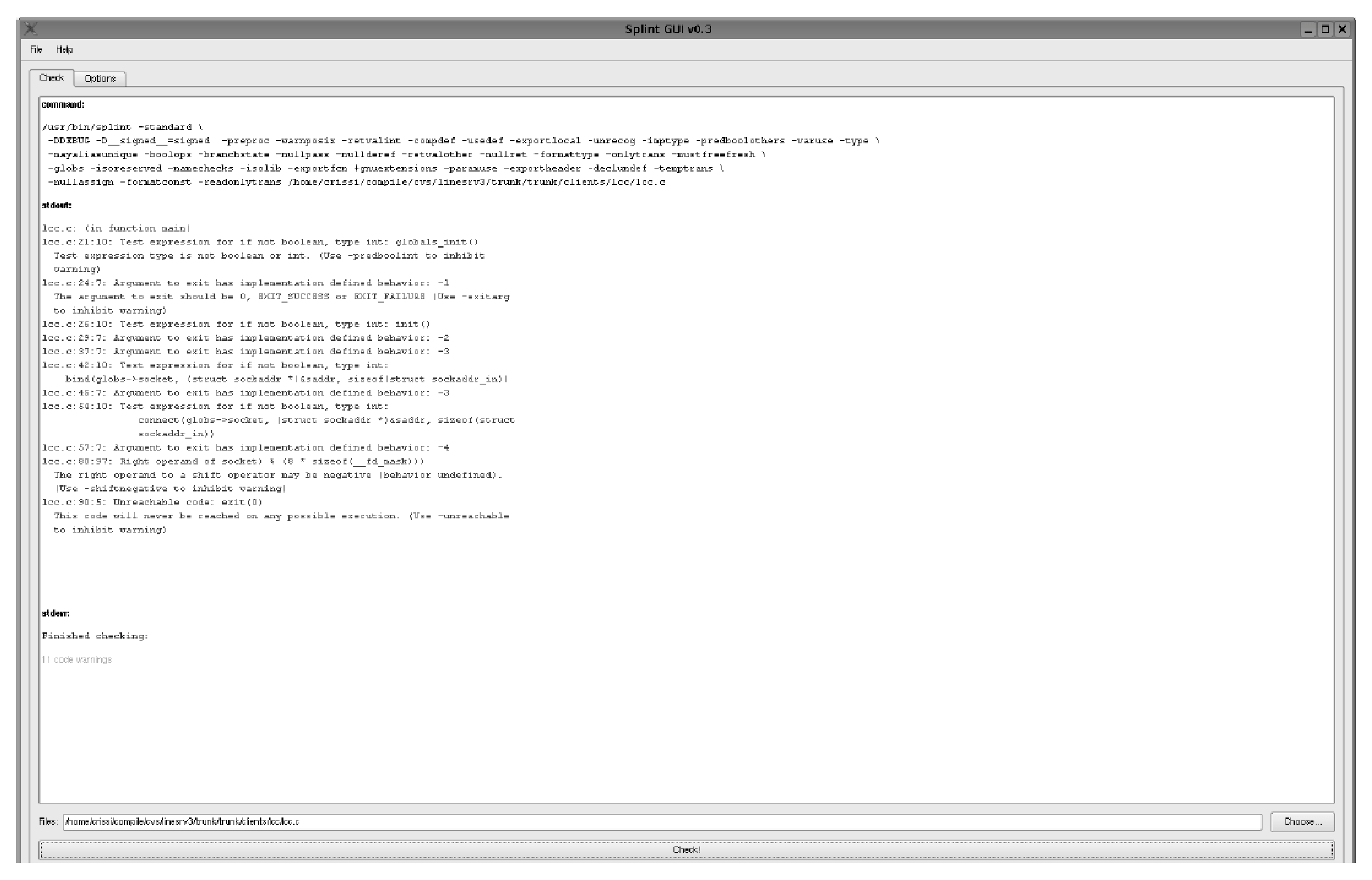


图 4-43　Splint 的检查界面

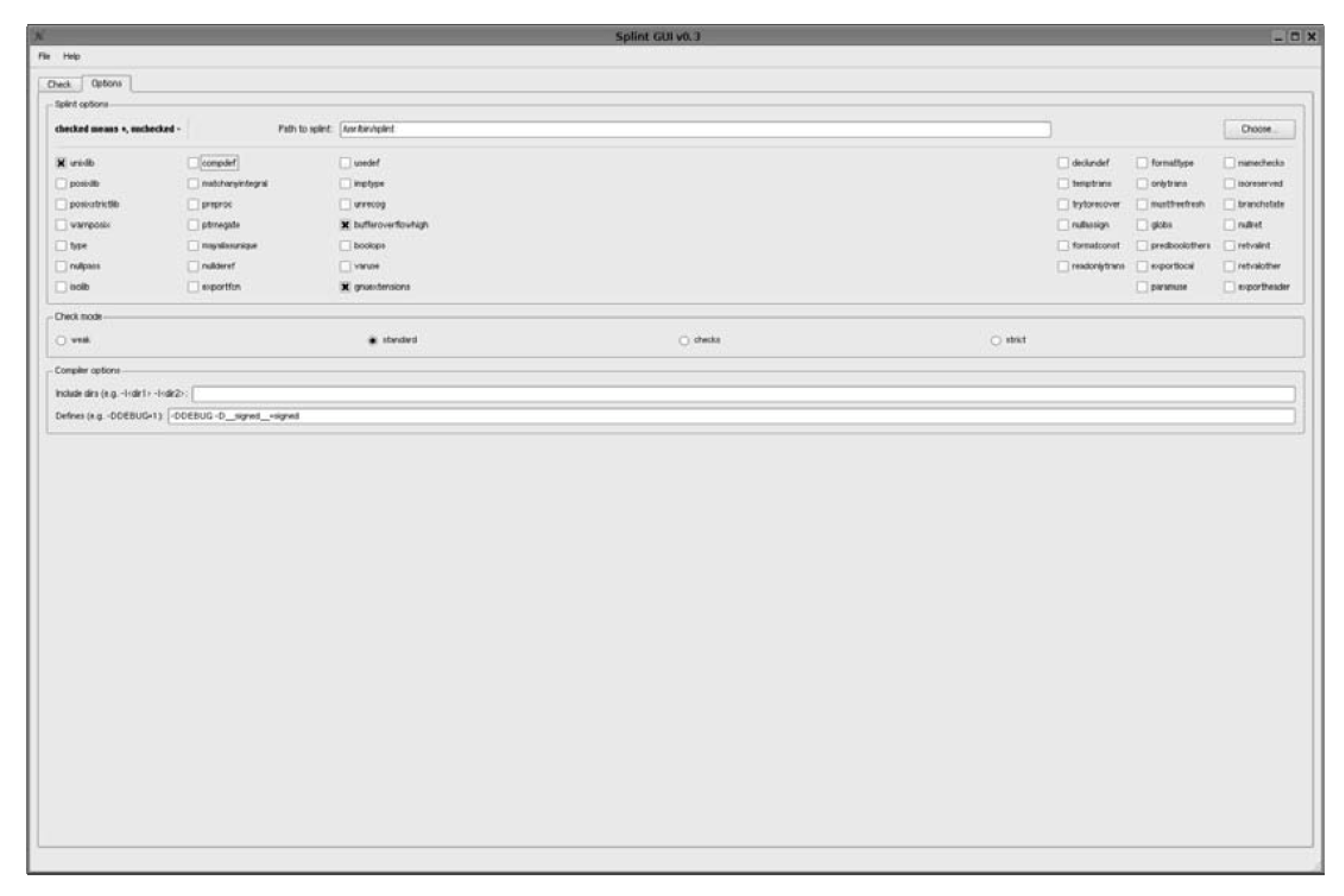


图 4-44　Splint 的配置界面

实 验 习 题

1. 分别应用 FindBugs、PMD 和 CheckStyle 对 Java 程序代码进行分析，并总结它们各自的特点。

2. 全面地应用 Valgrind、Prefast 或其他开源的静态分析工具，并给出它们完整的应用结果。

第III部分　单元测试篇

从软件测试的分类来看，如果从测试阶段的 V 模型考虑，单元测试是软件测试的基础，是软件测试中的第一个测试阶段。因此单元测试的效果会直接影响到软件的后期测试，最终在很大程度上影响到产品的质量。

从以下几个方面可以看出单元测试的重要性。

(1) 时间方面：认真做好单元测试，将会在系统集成联调时节约很多时间，反之由于各种原因放弃单元测试的做法，将会给后期集成测试和系统测试工作带来诸多隐患。

(2) 测试效果：单元测试是测试阶段的基础，能发现较深层次的问题，同时单元测试关注的范围相当特殊，它不仅关注程序代码解决何种问题，最重要的是关注代码如何解决问题。

(3) 测试成本：在单元测试阶段有些问题很容易被发现，可是若一直到后期的测试中才被发现，花费的成本将成倍上升。同理，定位问题和解决问题的费用也是成倍上升的，所以缺陷应该尽可能在早期被排除。

(4) 产品质量：单元测试完成的好坏直接影响到产品的质量等级，有时代码中的某一个小错误可能导致整个产品的质量降低一个指标，这些问题需要通过单元测试工作解决和避免。

综上所述，单元测试是构筑产品质量的基石，不要为了节约时间而放弃单元测试，这会在后期花费加倍的时间来弥补。任何软件开发团队都不愿意因为节约了早期单元测试的时间，导致开发的整个产品失败或重来。

由于目前软件代码的数量越来越多，一个软件程序中的软件单元也越来越多，单元中的代码也越来越复杂，单元测试的内容也比以前单元测试的要求更加全面。因此，完全利用手工进行软件单元测试需要增加更多的软件测试员和测试时间，并且会降低单元测试的覆盖率和准确度。同时，现代编程技术的发展也使软件单元测试自动化工具的实现成为可能，而这些自动化工具就是软件单元测试工具。

单元测试工具的一个重要功能就是测试的自动化，测试自动化的基础就是测试框架。现在许多编程语言都有相应的单元测试框架，目前最典型和最流行的单元测试框架是以 JUnit 测试框架为基础的 xUnit 测试框架。

xUnit 测试框架的主要功能就是对单元测试进行管理，并可进行自动化测试。xUnit 测试框架是在 1998 年作为 eXtreme 编程的核心概念引入。它提出了一个有效的机制，有助于开发人员将结构化、有效且自动的单元测试添加到常规开发活动中。从那以后，该框架演化为针对自动化单元测试框架的事实标准。

xUnit 根据语言不同分为 JUnit(Java)，CppUnit(C++)，DUnit (Delphi)，NUnit(.NET)，PhpUnit(PHP)等。

第5章　xUnit单元测试框架

众所周知，测试是软件开发中必不可少的一部分，是软件质量保障的重要手段。它有着自身完整的生命周期，贯穿软件开发过程的始终，相互影响。软件测试大致分为测试需求分析、测试计划设计、测试用例设计、执行测试这几个过程。在软件项目的每一个阶段都要进行不同目的和内容的测试活动，以保证各个阶段的正确性。软件测试的对象不仅包括软件代码，还包括软件需求文档和设计文档。而对软件代码的测试，又包括模块内部的单元测试、模块之间组合的集成测试以及编码结束后的系统测试、验收测试等。因此，每一级测试的效率和结果，都会直接影响到下一级的开发和测试过程。

单元测试作为代码级最小单位的测试，在开发过程中发挥着举足轻重的作用。它的作用是获取应用程序中可测软件的最小片段(函数甚至某条语句)，将其同其他代码隔离开来，然后确定它的行为同预期的一致。显然，只有保证了最小单位的代码准确，才能有效构建起基于它们之上的软件模块及软件系统。

极限编程(eXtreme Programming，XP)更加强调软件测试的重要性，而且非常注重单元测试这一环节。在使用 XP 进行软件开发时，如果没有单元测试，项目几乎就没有成功的机会。同时，它推崇测试优先原则，要求写代码之前就完成该部分的单元测试框架，使得程序员可以从测试的角度来考虑设计和代码实现，即测试驱动开发。

测试驱动开发(TDD)是以测试作为开发过程的中心。它坚持在编写实际代码之前，先写好基于产品代码的测试代码。开发过程的目标就是首先使测试能够通过，然后再优化设计结构。测试驱动开发是极限编程的重要组成部分，成功地应用 TDD 的一个潜在假设是有一个可用的单元测试框架。尽管商业工具是一个可选的方案，但大多数敏捷开发人员还是愿意使用 xUnit 系列的开源工具，比如 JUnit 或 VBUnit。如果没有这些工具，TDD 实际上是不可能的。

TDD 与敏捷开发不可分割，如果说敏捷开发是一条大船，那么 xUnit 测试框架就是这条船的发动机。xUnit 是各种代码驱动测试框架的统称，这些框架可以测试软件的不同内容(单元)，比如函数和类。xUnit 框架的主要优点是，它提供了一个自动化测试的解决方案。没有必要多次编写重复的测试代码，也不必记住这个测试的结果应该是怎样的。

5.1　xUnit 介绍

xUnit 源于 Kent Beck 曾经为 Smalltalk 编写的测试框架——SUnit。后来 Kent Beck 和 Erich Gamma 一起将这个框架移植到了 Java 上，被称为“JUnit”。自 1999 年以来，JUnit 已经发展成业界标准的 Java 测试和设计工具，获得了广泛的应用，不仅在开源的项目(www.opensource.org)中使用，还经常被商业软件公司所使用。Kent Beck 的框架结构背后所表现的一些概念，被抽象成 xUnit，并慢慢演化成一些简单的编写测试的规则。目前已有一套完善的测试工具和方法论来支持了，适用于各种语言的单元测试。例如，它的框架被移植到三十多种语言和环境中，即 xUnit 有很多的成员，如 JUnit、PythonUnit、CppUnit、NUnit 等。

1. xUnit 测试框架

xUnit 是一个基于测试驱动开发的单元测试框架。单元测试框架的主要目标是提供编写、运行测试用例，反馈测试结果以及记录测试日志的一系列基础软件设施。它为用户在开发过程中使用测试驱动开发提供了一个方便的工具，使用户得以快速地进行单元测试。

xUnit 的测试框架非常简单和实用，如图 5-1 所示。该框架支持自动化测试，能够捕获异常、记录错误和失败，并能利用组合模式对 Test Case 和 Test Suite 进行管理。实际上，还有很多工作是可以在 xUnit 框架上继续开展的，例如，软件开发中是不是存在较为通用的测试用例？如果是，就可以定义一些抽象的测试用例，并以此作为测试框架的基础。

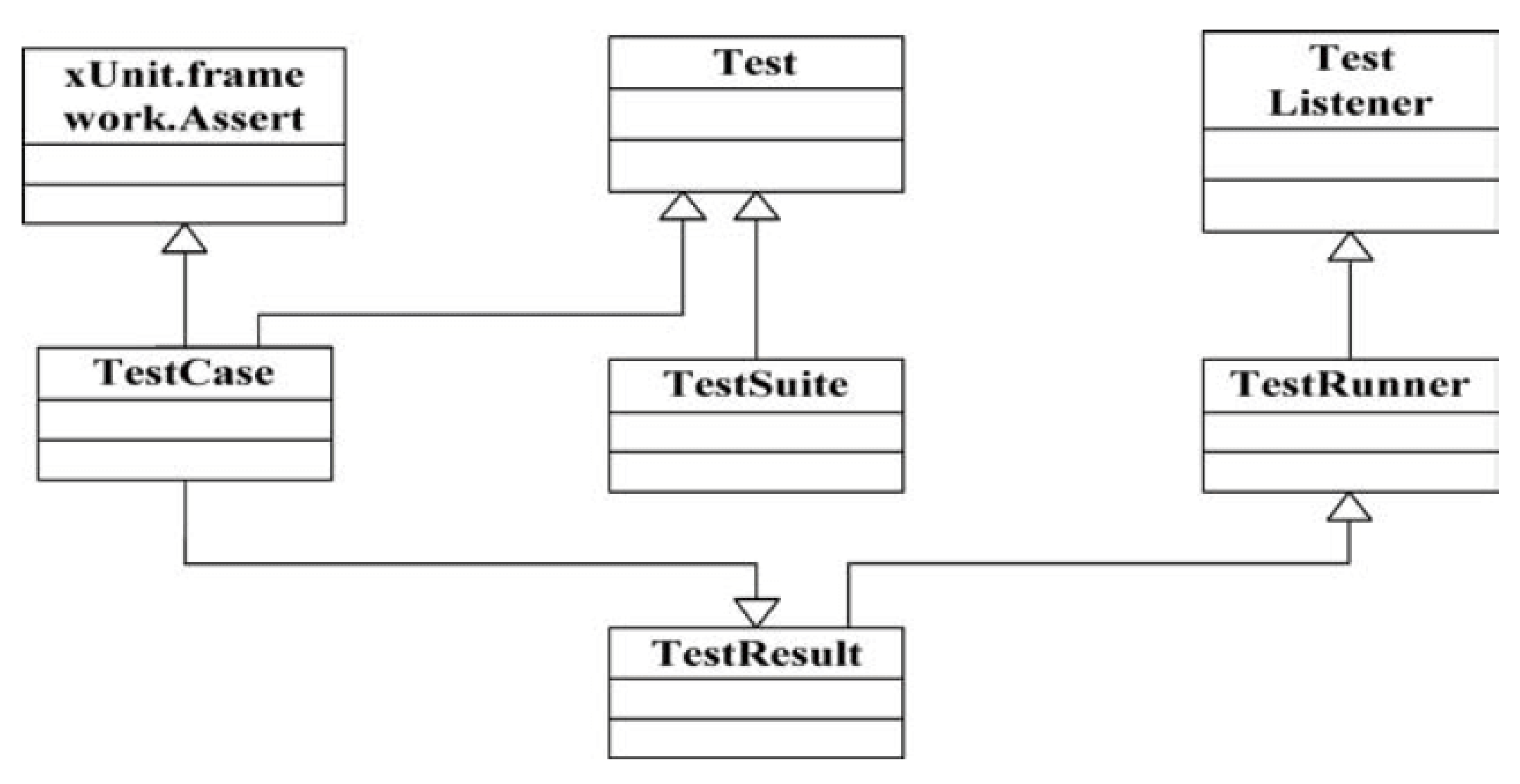


图 5-1　xUnit 测试框架

下面应用极限编程开发方法，首先编写测试代码，然后编写符合测试的代码，进而完成整个软件。为此，可以总结出平时开发过程中单元测试的几条原则。

(1) 先编写测试代码，然后编写符合测试的代码。至少做到完成部分代码后，再完成对应的测试代码。

(2) 测试代码不需要覆盖所有的细节，但应该对所有主要的功能和可能出错的地方有相应的测试用例。

(3) 发现 Bug 后，首先应编写对应的测试用例，然后再进行调试。

(4) 不断总结出现 Bug 的原因，对其他代码也编写相应的测试用例。

(5) 每次编写完代码后，运行所有以前的测试用例，验证对以前代码的影响，并把这种影响尽早消除。

(6) 不断维护测试代码，保证代码变动后能通过所有测试。

为了保证上述原则得以实施，利用 xUnit 测试框架来帮助管理测试代码，完成自动测试。

2. xUnit 测试框架 4 大要素

xUnit 测试框架包括 4 大要素：测试 Fixtures、测试集、测试执行和测试断言。

1) 测试 Fixtures

测试 Fixtures 是一组认定被测对象或被测程序单元测试成功的预定条件或预期结果的设定。Fixture 就是被测试的目标，可能是一个对象或一组相关的对象，甚至是一个函数。测试人员在测试前就应该清楚对被测对象进行测试的正确结果是什么，这样就可以对测试结果有一个明确判断。

2) 测试集

测试集就是一组测试用例，这些测试用例要求有相同的测试 Fixtures，以保证这些测试不会出现管理上的混乱。

3) 测试执行

单个单元测试的执行可以按下面的方式进行：

```
setUp(); /* 首先，要建立针对被测程序单元的独立测试环境 */
...
/* 然后，编写所有测试用例的测试体/测试程序 */
...
tearDown(); /* 最后，无论测试成功还是失败，都要将被测程序的测试环境进行清理，以免影响后继的测试*/
```

setUp() 和 tearDown()方法用于测试 Fixtures 的初始化和测试完成后的现场清理。

4) 断言

断言实际上就是验证被测程序在测试中的行为或状态的一个宏或函数。断言失败实际上就是引发异常，终止测试的执行。

3. xUnit 的测试流程

有了被测试的 Fixture，就可以对这个 Fixture 的某个功能、某个可能出错的流程编写测试代码，这样对某个方面完整的测试被称为 TestCase(测试用例)。通常写一个 TestCase 的步骤如下。

(1) 对 Fixture 进行初始化，以及其他初始化操作。比如，生成一组被测试的对象，初始化值。

(2) 按照要测试的某个功能或流程对 Fixture 进行操作。

(3) 验证结果是否正确。

(4) 对 Fixture 的以及其他的资源释放等做清理工作。

对 Fixture 的多个测试用例，通常步骤(1)、(2)部分的代码都是相似的，xUnit 在很多地方引入了 setUp()和 tearDown()虚函数。可以在 setUp()函数里完成步骤(1)中的初始化代码，而在 tearDown()函数中完成步骤(4)的代码。具体测试用例函数中只需要完成步骤(2)、(3)部分的代码即可，运行时 xUnit 会自动为每个测试用例函数运行 setUp()，之后运行 tearDown()，这样测试用例之间就不会产生交叉影响。

对 Fixture 的所有测试用例可以被封装在一个 xUnit::TestFixture 的子类(命名惯例是[ClassName]Test)中。然后定义这个 Fixture 的 setUp()和 tearDown()函数，为每个测试用例定义一个测试函数(命名惯例是 test×××)。

5.2　JUnit 单元测试工具

1997 年，Erich Gamma 和 Kent Beck 为 Java 语言创建了一个简单但有效的单元测试框架，称作 JUnit。这项工作遵循 Kent Beck 在早些时候为 Smalltalk 创建的名为 SUnit 测试框架的设计。

JUnit 是一个 SourceForge 上的开源软件，以 IBM 的 Common Public License 1.0 版授权协议发布。Common Public License 授权协议对商业用户是友好的，人们可以随同产品分发 JUnit，不需要很多官样文件，也没有很多限制。

JUnit 很快成了 Java 开发中单元测试框架的事实标准。事实上，基于 JUnit 的测试模型(即 xUnit)已正式成为各种语言的标准框架。ASP、C++、C#、Effel、Delphi、Perl、PHP、Python、REBOL、Smalltalk 和 Visual Basic 都已经有了 xUnit 框架。

那么 JUnit 的目标是什么呢？首先，回到开发的前提假设。假设如果一个程序不能自动测试，那么它就不会工作。但有更多的假设认为，如果开发人员保证程序能工作，那么它就会永远正常工作。与这个假设相比，我们的假设实在是太保守了。从这个观点出发，开发人员编写了代码，并进行了调试，还不能说他的工作就完成了，他还必须编写测试脚本，以证明程序工作正常。然而，每个人都很忙，没有时间去进行测试工作。他们会说，

我编写程序代码的时间都很紧，哪儿有时间去写测试代码呢？ 因此，首要的目标就是构建一个测试框架，在这个框架里，开发人员能编写测试代码。框架要使用熟悉的工具，无须花很多精力就可以掌握。它还要消除不必要的代码，除了必需的测试代码外，消除重复劳动。

如果测试要做的仅仅是这些，那么在调试器中编写一个表达式就可以实现。但是，测试不仅这些。虽然程序工作得很好，但这不够，因为不能保证集成后程序还能正常工作。因此，测试的第二个目标就是创建测试，并能保留这些测试，将来它们也是有价值的，其他人可以执行这些测试，并验证测试结果。如果有可能，还要把不同人的测试收集在一起，一起执行，且不用担心它们之间会互相干扰。最后，还要能用已有的测试创建新的测试。每次创建新的测试 Fixture 是很花费代价的，框架能复用测试设置，执行不同的测试。

使用 JUnit 的好处有：

(1) 可以使测试代码与产品代码分开。

(2) 针对某一个类的测试代码，通过较少的改动便可以将其应用于另一个类的测试。

(3) 易于集成到测试人员的构建过程中，JUnit 和 Ant 的结合可以实施增量开发。

(4) JUnit 是公开源代码的，可以进行二次开发。

(5) 可以方便地对 JUnit 进行扩展。

JUnit 测试编写原则：

(1) 简化测试的编写，这种简化包括测试框架的学习和实际测试单元的编写。

(2) 使测试单元保持持久性。

(3) 可以利用既有的测试来编写相关的测试。

JUnit 的特征：

(1) 使用断言方法来判断期望值和实际值的差异，返回 Boolean 值。

(2) 测试驱动设备使用共同的初始化变量或实例。

(3) 测试包结构便于组织和集成运行。

(4) 支持图形交互模式和文本交互模式。

JUnit 共有 7 个包，核心的包就是 junit.framework 和 junit.runner。Framework 包负责整个测试对象的构架，Runner 包负责测试驱动。

JUnit 有 4 个重要的类：TestSuite、TestCase、TestResult 和 TestRunner。前三个类属于 Framework 包，后一个类在不同的环境下是不同的。如果使用的是文本测试环境，这时用的就是 junit.textui.TestRunner。各个类的职责如下。

(1) TestResult：负责收集 TestCase 所执行的结果。它将结果分为两类：客户可预测的 Failure 和没有预测的 Error。同时负责将测试结果转发到 TestListener(该接口由 TestRunner 继承)进行处理。

(2) TestRunner：客户对象调用的起点，负责对整个测试流程进行跟踪。能够显示返回的测试结果，并且报告测试的进度。

(3) TestSuite：负责包装和运行所有的 TestCase。

(4) TestCase：客户测试类所要继承的类，负责测试时对客户类进行初始化以及测试方法调用。

另外还有两个重要的接口：Test 和 TestListener。

(1) Test：包含两个方法，run()和 countTestCases()，用于对测试动作特征的提取。

(2) TestListener：包含 4 个方法，addError()、addFailure()、startTest()和 endTest()，用于对测试结果的处理以及测试驱动过程的动作特征的提取。

JUnit 框架的组成：

(1) 对测试目标进行测试的方法与过程集合，可称为测试用例(TestCase)。

(2) 测试用例的集合，可容纳多个测试用例(TestCase)，将其称作测试包(TestSuite)。

(3) 测试结果的描述与记录(TestResult)。

(4) 测试过程中的事件监听者(TestListener)。

(5) 每一个测试方法所发生的与预期不一致的状况的描述，称其为测试失败元素(TestFailure)。

(6) JUnit Framework 中的出错异常(AssertionFailedError)。

(7) JUnit 框架是一个典型的 Composite 模式：TestSuite 可以容纳任何派生自 Test 的对象；当调用 TestSuite 对象的 run()方法时，会遍历自己容纳的对象，逐个调用它们的 run()方法。

典型的使用 JUnit 的方法就是继承 TestCase 类，然后重载它的一些重要方法：setUp()、tearDown()、runTest()(这些都是可选的)。最后再将这些客户对象组装到一个 TestSuite 对象中，交由 junit.textui.TestRunner.run (用例集)驱动。

5.2.1　JUnit 单元测试环境建立

由于 JUnit 越来越流行，现在很多 Java 的 IDE 都会集成 JUnit 插件，使用起来也非常方便。虽然现在绝大部分开发人员都是用 IDE 工具来开发程序，但是为了能够让更多的人体验 JUnit 而不必去下载体积庞大的 IDE 工具，本节将分两种情况介绍 JUnit 测试环境的建立。一种是最直接的方式：配置 JUnit，通过命令行来建立。另一种则是使用 Eclipse 中的 JUnit 插件来建立。

注意：使用 JUnit 和 IDE 的前提是系统中已安装 Java 的 JDK(Java Develope Kit)，并正确加入了系统环境变量。

1. 独立 JUnit 测试环境的建立

(1) 从 www.junit.org(JUnit 官方网站)下载最新的 JUnit 包(这里使用的是 JUnit 3.8.1 版本)，如图 5-2 所示。



图 5-2　JUnit 官方网站下载页面

(2) 将 JUnit 的压缩包解压到硬盘上(这里是解压到 C 盘)，如图 5-3 所示。



图 5-3　将 JUnit 解压到本地硬盘

(3) 将 JUnit 的 jar 包(这里的示例为 junit.jar，JUnit 版本不同，该 jar 包的名字也会不同)添加到环境变量的 classpath(默认的 class 文件路径变量)中。

(4) 右击“我的电脑”，依次选择“属性”|“高级”|“环境变量”。如果系统环境变量中没有 classpath 这个变量，则需要手动添加一个。单击“系统变量”下的“新建”按钮，如图 5-4 所示。

在打开的对话框中按如图 5-5 所示的信息填写。



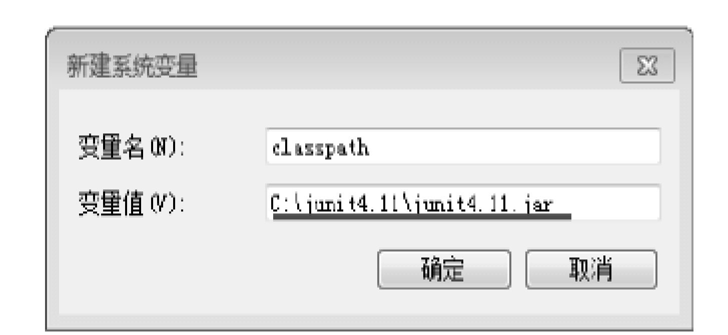



图 5-4　“环境变量”对话框

图 5-5　设置环境变量

注意：图中下划线标注部分会根据 JUnit 安装路径和版本的不同而变化，请做相应更改。

(5) 测试安装是否成功。

有以下三种方式可以用来进行测试。

① 批处理文本方式，在 cmd 命令行中输入如图 5-6 所示的命令。



图 5-6　命令行测试

② AWT 图形测试运行方式，在 cmd 命令行中输入如图 5-7 所示的命令。

③ 基于 Swing 的图形测试方式，在 cmd 命令行中输入如图 5-8 所示的命令。

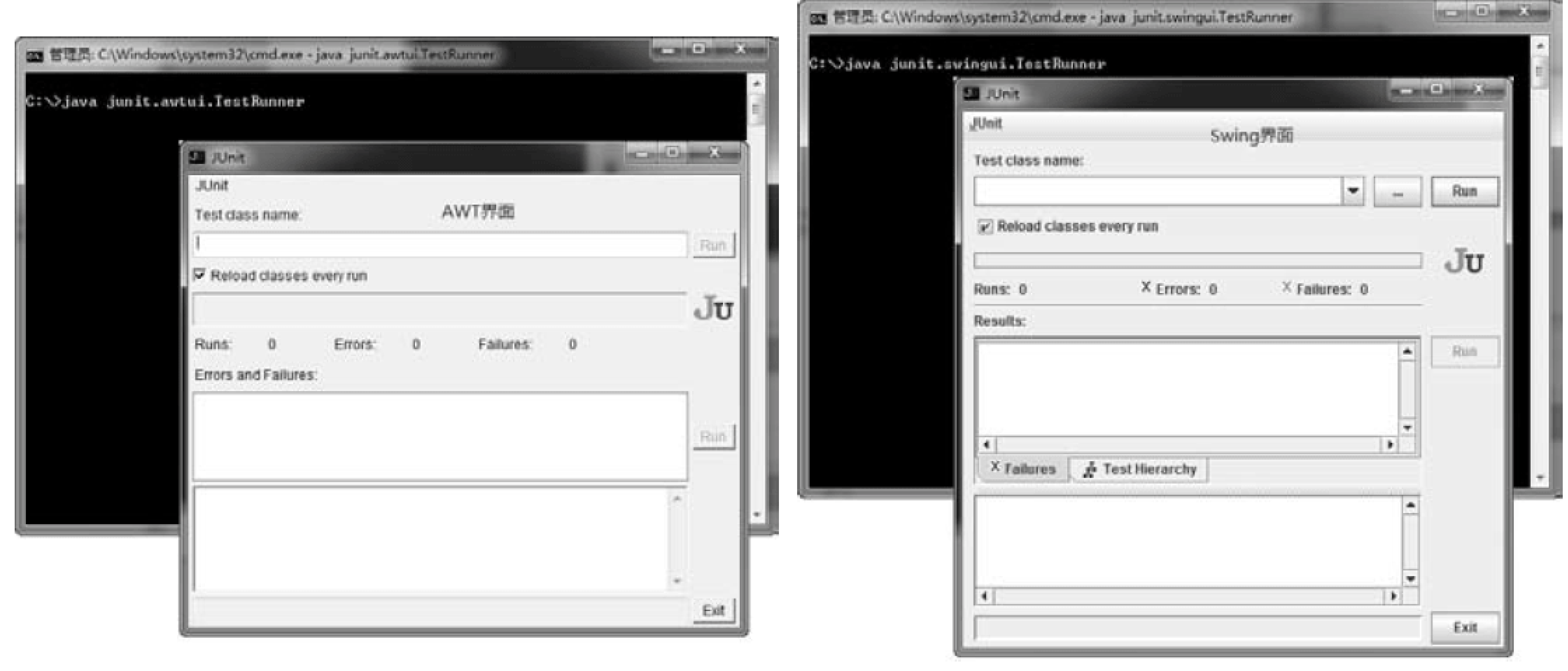


图 5-7　AWT 图形测试　　　　图 5-8　Swing 图形测试

如果上述命令都运行无误，则安装成功。

2. Eclipse 中的 JUnit 插件

由于 Eclipse 中已经集成了 JUnit 插件，所以可以直接建立测试用例来测试代码。即使所使用的 Eclipse 中没有集成 JUnit，也没有关系。不需要特别的配置，只要在工程属性中引入 JUnit 的 jar 包即可使用。

1) Eclipse 中包含 JUnit4 插件

此时，只需要在工程中应用即可，应用方法如下。

(1) 右击 Java 工程，选择“属性”命令，打开如图 5-9 所示对话框，并选择 Java Build Path 选项。

(2) 在 Libraries 选项卡中单击 Add Library 按钮，选择 JUnit4，如图 5-10 所示。

(3) 单击 Finish 按钮，在工程目录下即可看见 JUnit4 lib 包，如图 5-11 所示。

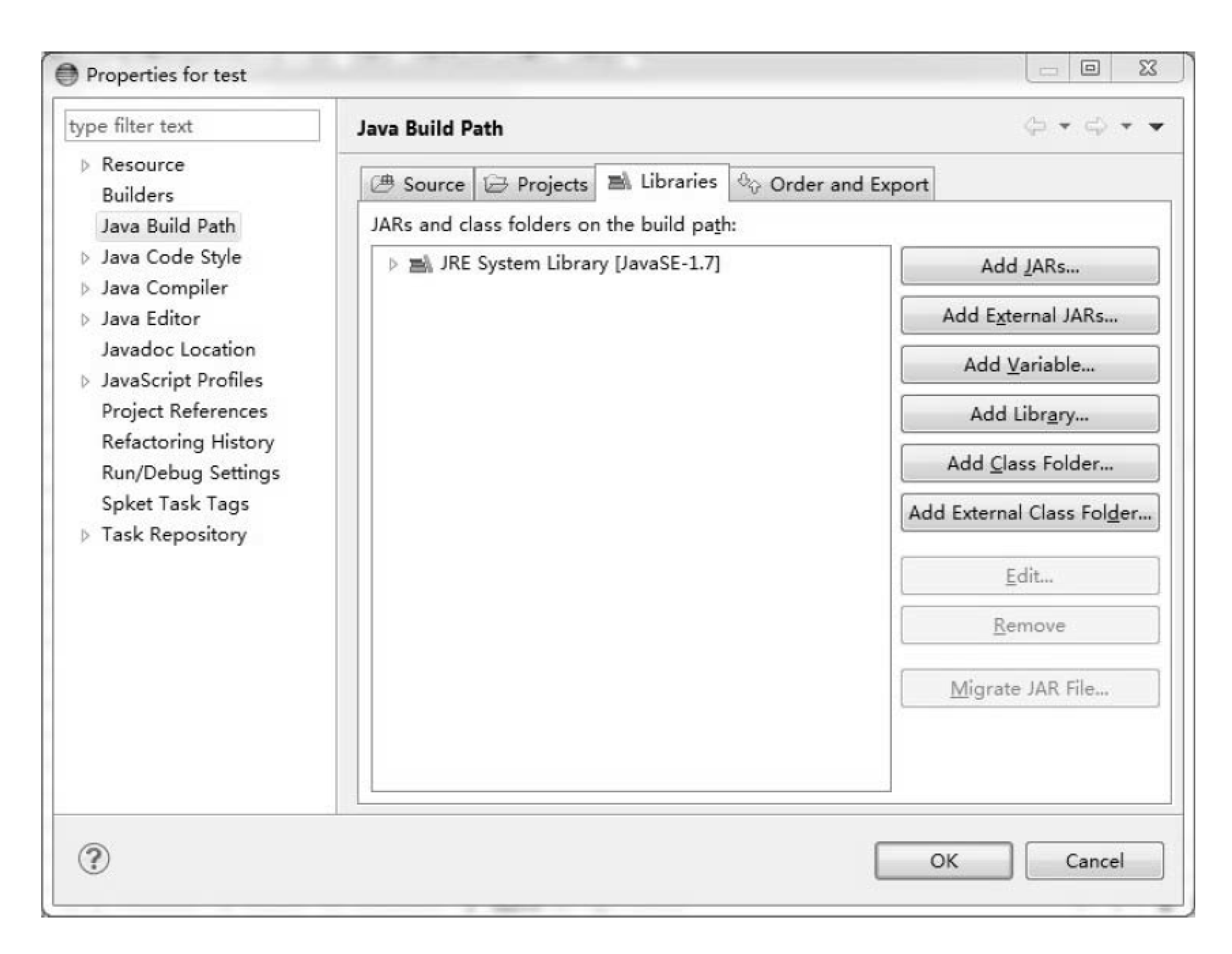


图 5-9　选择 Java Build Path

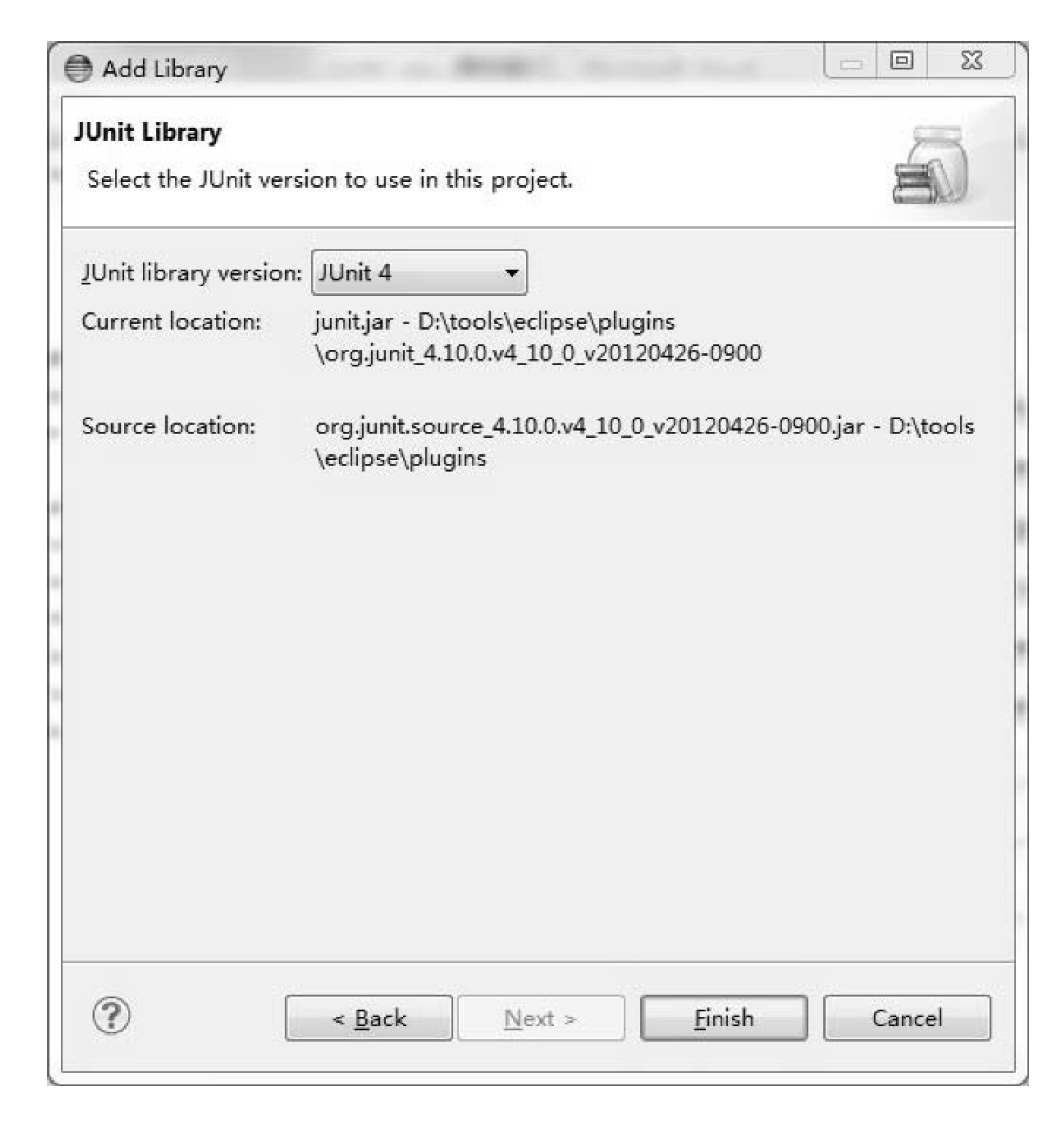


图 5-10　选择 JUnit4

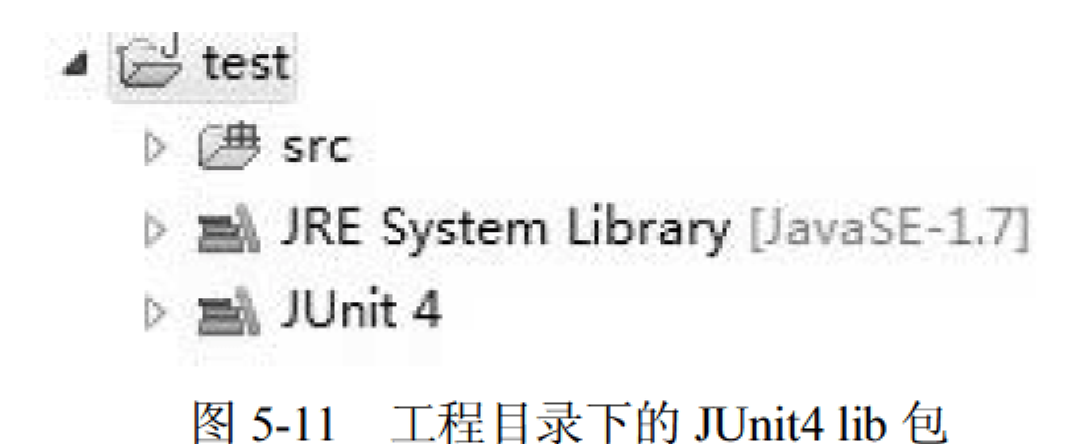


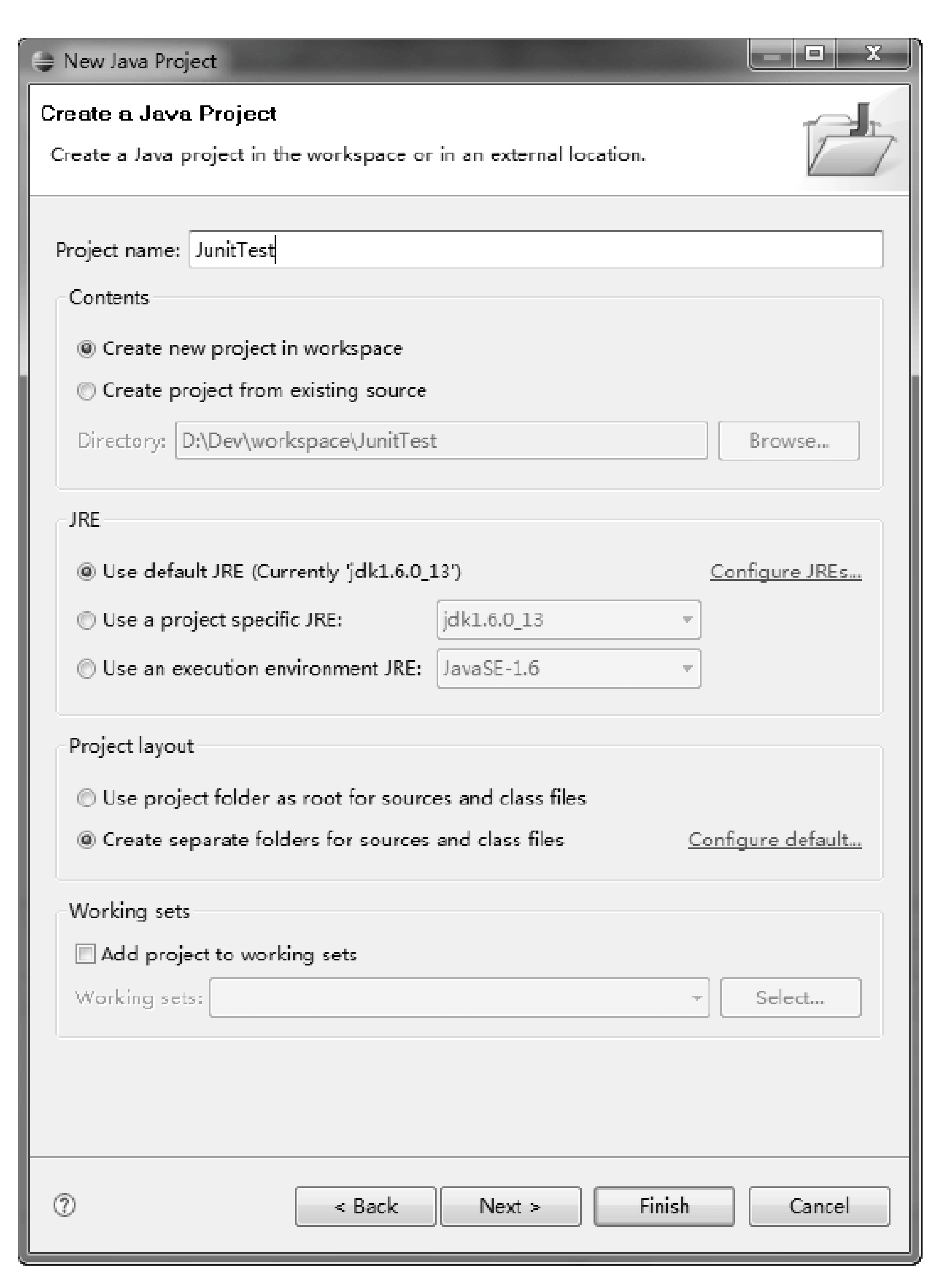
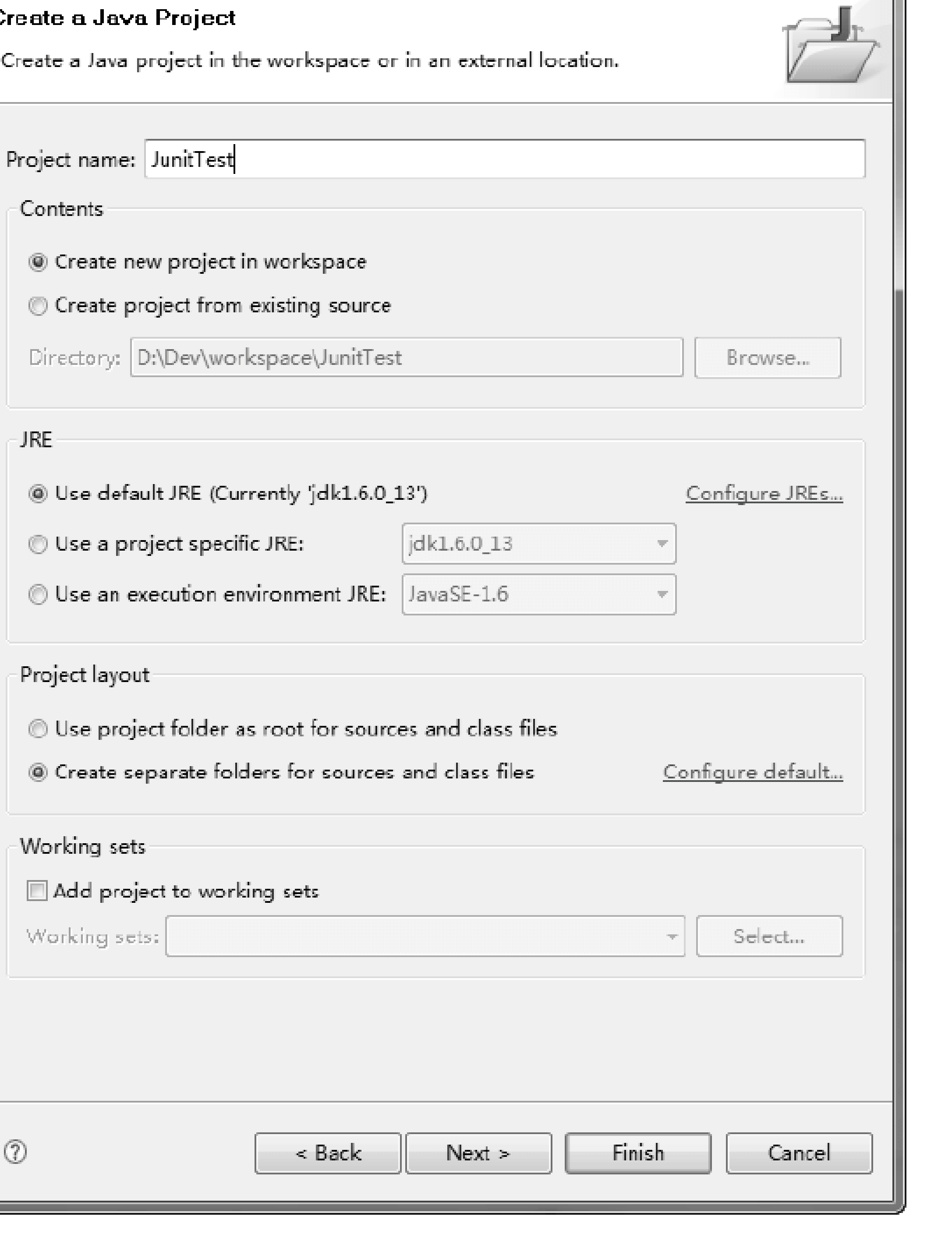
图 5-11　工程目录下的 JUnit4 lib 包

2) Eclipse 中未包含 JUnit4 插件

如果 Eclipse 中没有集成 JUnit 插件，按以下步骤就可以轻松集成这个强大的测试框架。

(1) 下载 JUnit 的集成包，并将其解压缩到硬盘上(方法与前面介绍的一样，这里不再赘述)。然后在 Eclipse 中创建一个名为 JunitTest 的 Java 项目，如图 5-12 所示。

(2) 在 Package Explorer 中，右击刚才创建的项目名称，在弹出的菜单中选择 Properties 命令，如图 5-13 所示。

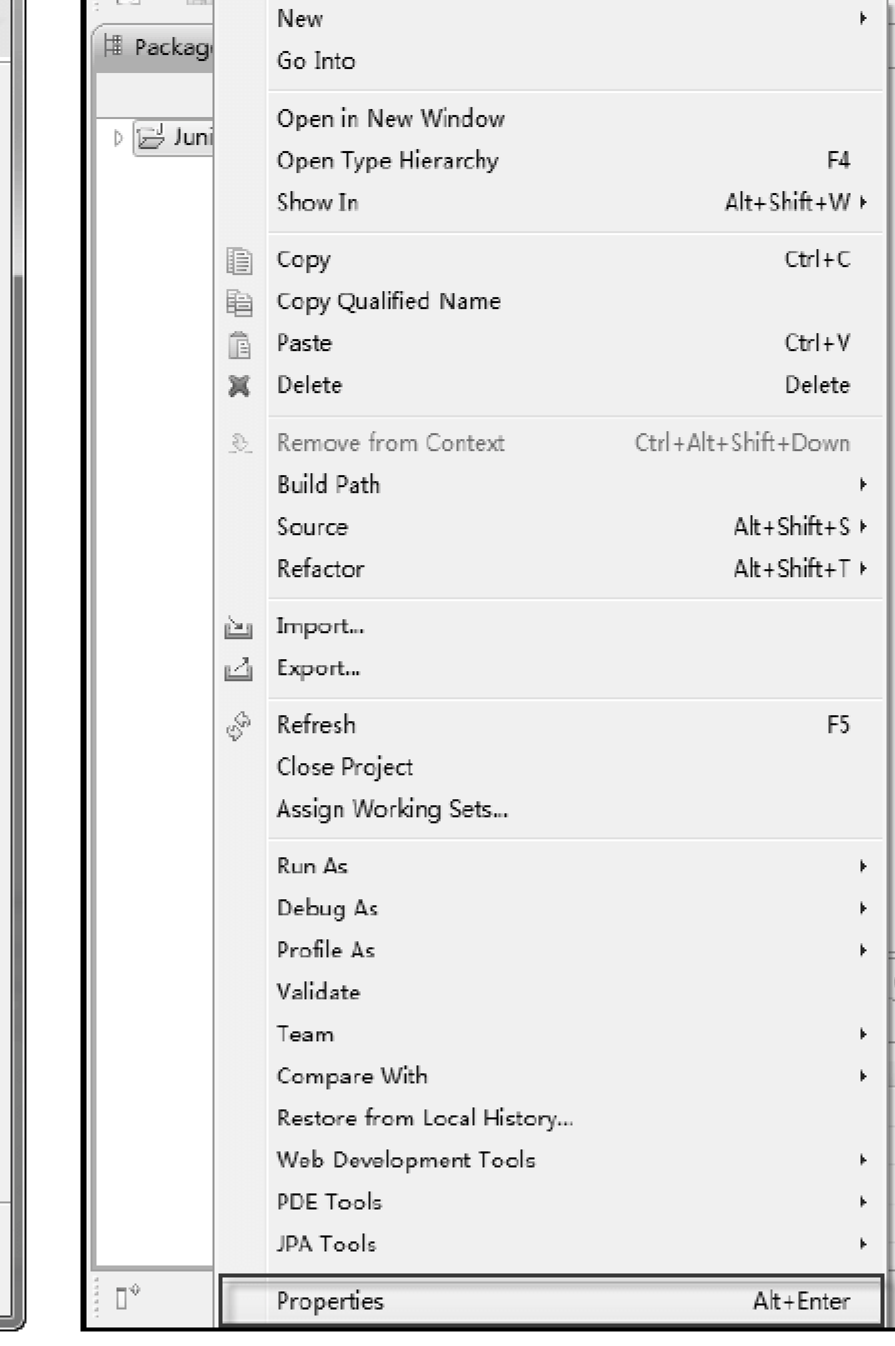


图 5-12　创建 Java 项目　　　　图 5-13　设置 Java 项目

(3) 依次选择 Java Build Path|Libraries，单击 Add External JARs 按钮，导航至 JUnit 解压缩的目录，选择 junit.jar 包，打开即可，如图 5-14 所示。

(4) 随便建立一个 Java 文件，右击这个文件，在菜单中选择 New 命令，这时候里面会有一个 JUnit Test Case 选项，单击它，就可以创建 JUnit 测试用例了，如图 5-15 所示。

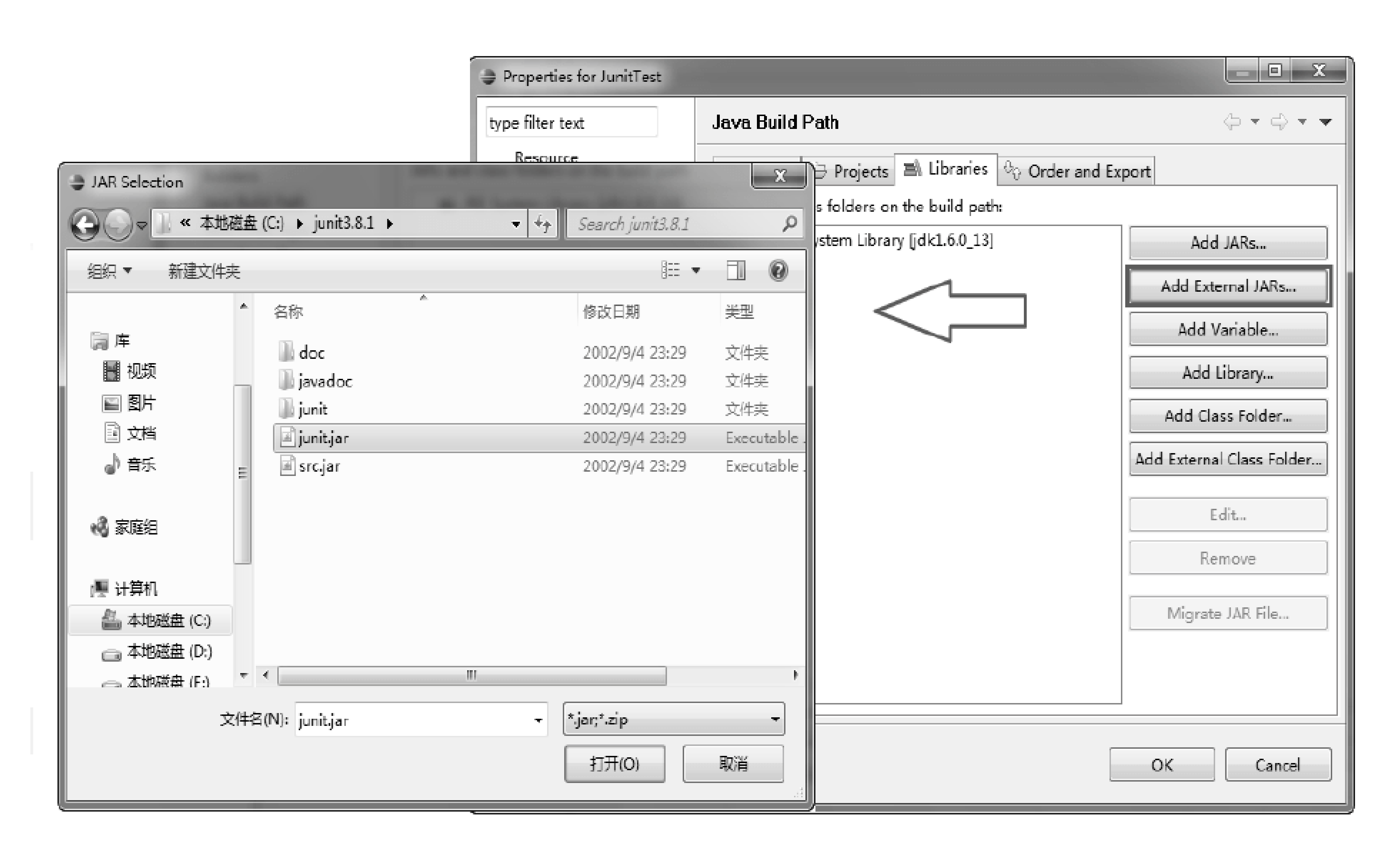


图 5-14　添加 JUnit 支持

JUnit 会根据选择的文件，自动把测试用例需要的参数填充完整。当然，如果不是根据文件创建的测试用例，而是完全自己手工写，要注意把所需的参数填齐全。

如果 Eclipse 中已经集成了 JUnit，那么根据步骤(4)的介绍直接使用 JUnit 即可。

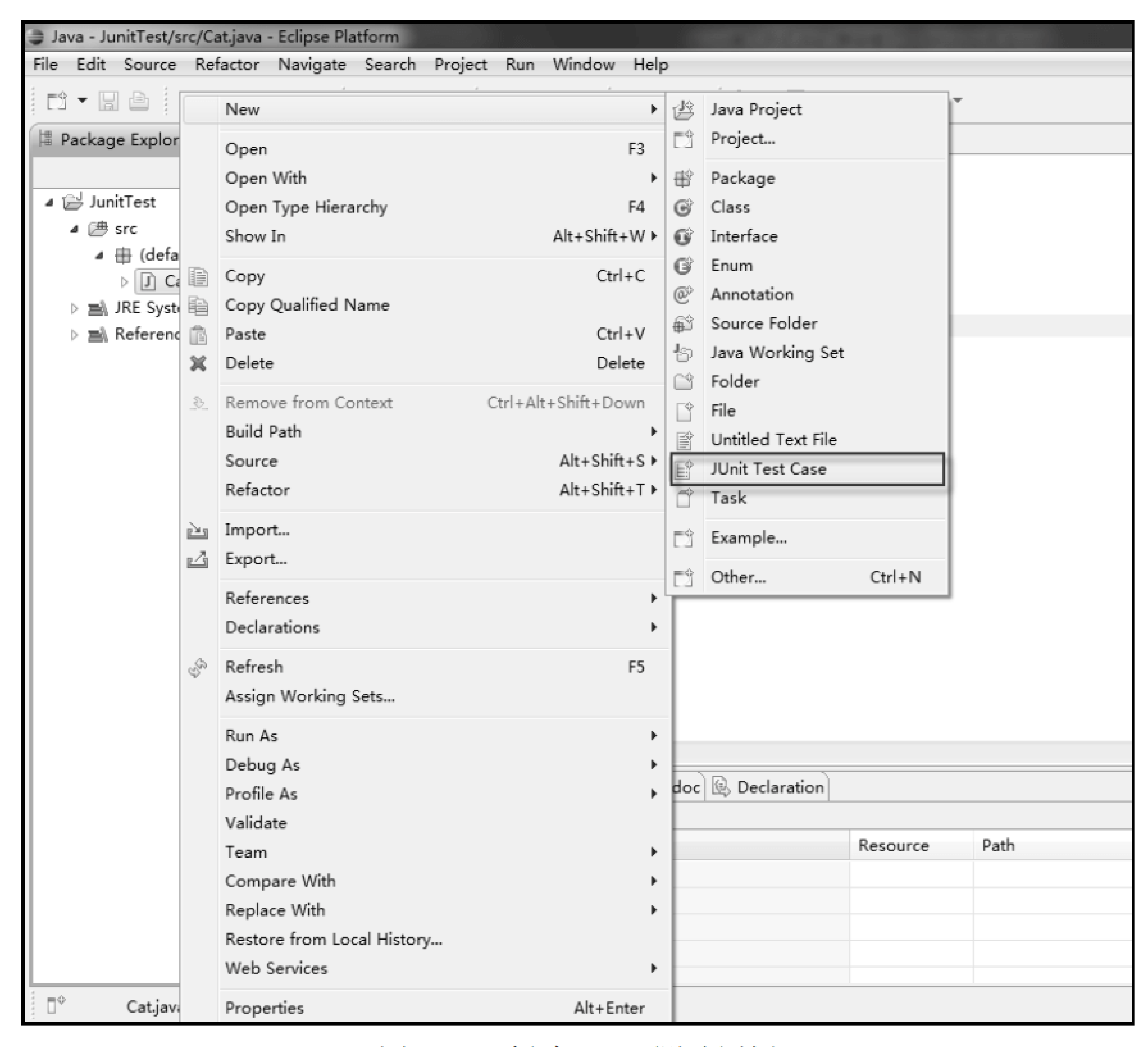


图 5-15　创建 JUnit 测试用例



图 5-15　(续)

5.2.2　JUnit 单元测试方法

接下来，可能会先编写测试代码，再编写工作代码；或者相反，先编写工作代码，再编写测试代码。按照 XP 编程开发方法，则应先编写测试代码，再编写工作代码。因为这样可以在编写工作代码时清晰地了解工作类的行为。

要注意编写那些能通过的测试的测试代码意义不是十分突出，而那些能帮助发现 Bug 的测试代码才有其价值。此外，测试代码还应该对工作代码进行全面的测试。

根据上面介绍 JUnit 测试环境搭建的内容，这里也分两种情况介绍其测试方法。

1. 独立 JUnit 应用

(1) 创建一个简单的 Java 类，存放于 C 盘 JunitTest 目录下。类的代码为：

```
public class Cat {
        public String getName(){
            return "Hello Kitty";
        }
 }
```

(2) 创建该类的测试类，放于同一个目录下。类的代码为：

```
import junit.framework.*;
  public class TestCat extends TestCase {
  protected String expectedLegs;
  protected Cat myCat;
```

```
    public TestCat(String name) {
        super(name);
    }
    // 设定了进行初始化的任务
    protected void setUp() {
        expectedLegs = "Hello Kitty";
        myCat = new Cat();
    }
    // 这是一个很特殊的静态方法。JUnit 的 TestRunner 会调用 suite 方法来确定有多少个测试可以执行
    public static Test suite() {
    return new TestSuite(TestCat.class);
    }
   // 对预期的值和 myCat.getLegs()返回的值进行比较，并打印比较结果
    public void testGetLegs() {
        assertEquals(expectedLegs, myCat.getName());
    }
}
```

(3) 编译测试类。

(4) 输入如图 5-16 所示的命令来执行 JUnit 测试。

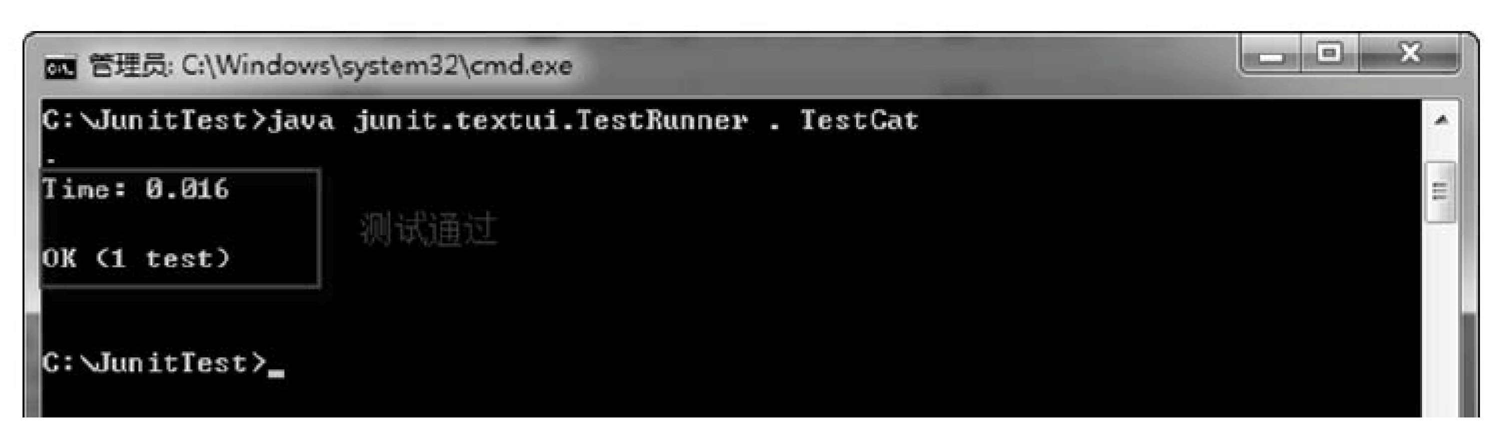


图 5-16　执行 JUnit 测试用例(命令行模式)

如果想启动 Swing 或 AWT 的 JUnit 界面来执行测试，则需要输入命令“java junit. swingui.TestRunner . TestCat”或“java junit.awtui.TestRunner.TestCat”，如图 5-17 所示。

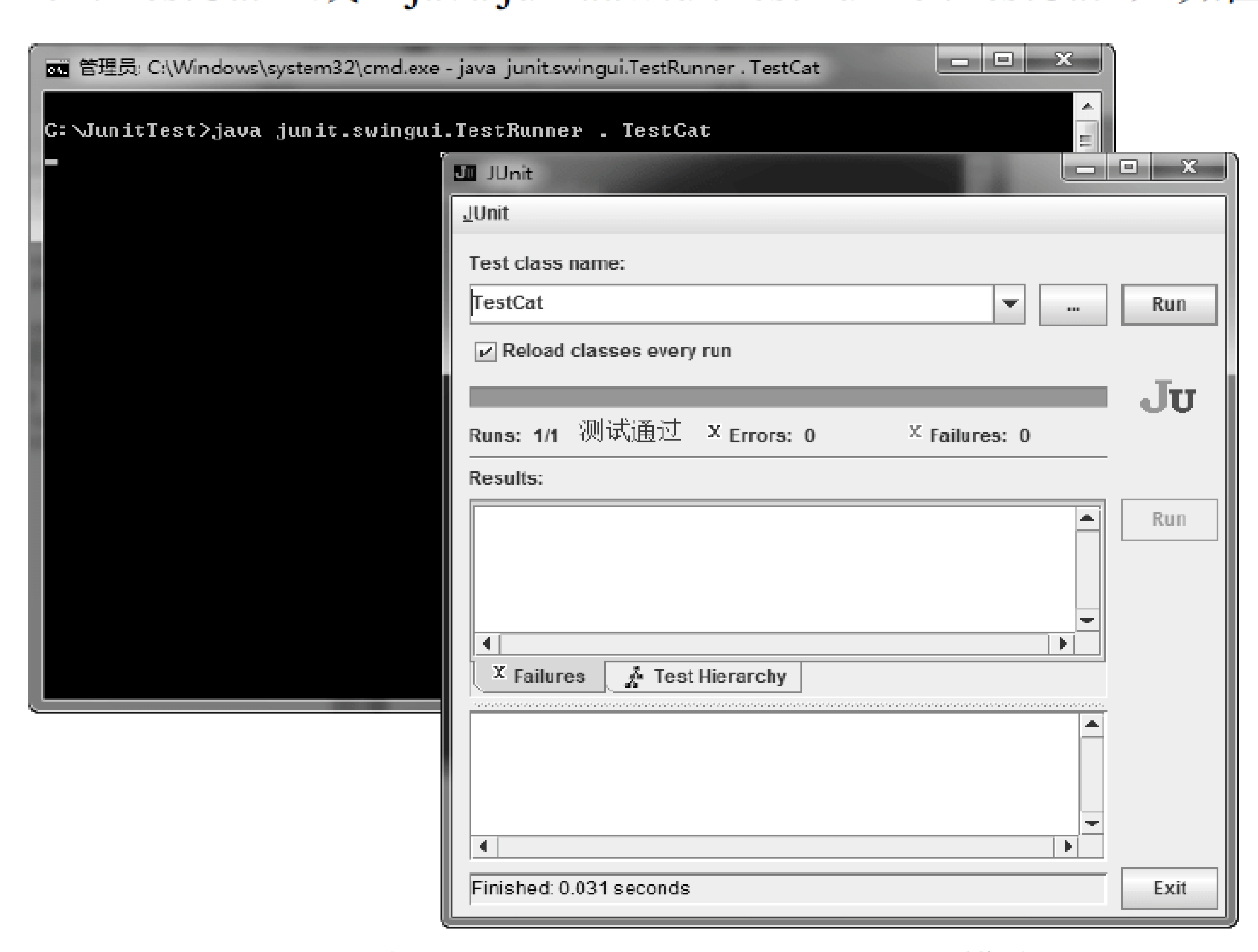


图 5-17　执行 JUnit 测试用例(Swing 界面模式)

2. Eclipse 中的 JUnit 应用

(1) 创建一个新的 Java 项目，然后创建一个简单的 Java 类，如图 5-18 所示。

Cat.java　TestCat.java

```
public class Cat {

    public String getName(){
        return "Hello Kitty";
    }
}
```

图 5-18　创建 Cat 类

(2) 按照前面介绍的方式，为 Cat 创建一个 JUnit 测试类，如图 5-19 所示。注意图中下划线标注的代码被修改过，这是为了测试 JUnit 测试不通过的情况。

Cat.java　TestCat.java

```
import junit.framework.*;

public class TestCat extends TestCase {

    protected int expectedLegs;

    protected Cat myCat;

    public TestCat(String name) {
        super(name);
    }

    // 设定了进行初始化的任务
    protected void setUp() {
        expectedLegs = 4;     实际应为String类型的"Hello Kitty"
        myCat = new Cat();
    }

    // 是一个很特殊的静态方法。JUnit的TestRunner会调用suite方法来确定有多少个测试可以
    public static Test suite() {

        return new TestSuite(TestCat.class);
    }

    // 对预期的值和myCat.getLegs()返回的值进行比较，并打印比较的结果
    public void testGetLegs() {
```

图 5-19　Cat 类的测试类 TestCat

(3) 依次选择 Eclipse 菜单栏中的 Run|Run as|JUnit Test 命令，执行 JUnit 测试。由于修改了断言中的内容，所以 JUnit 测试应该是通不过的，如图 5-20 所示。

把 TestCat 类中的代码修改为正确的，再执行 JUnit 测试，结果是可以通过，如图 5-21 所示。

5.2.3　JUnit 单元测试应用举例

1. 借用经典的售货机 Java 例子来说明 JUnit 测试过程

(1) 若投入 5 角钱或 1 元钱的硬币，按下“橙汁”或“啤酒”按钮，则相应的饮料就送出来。

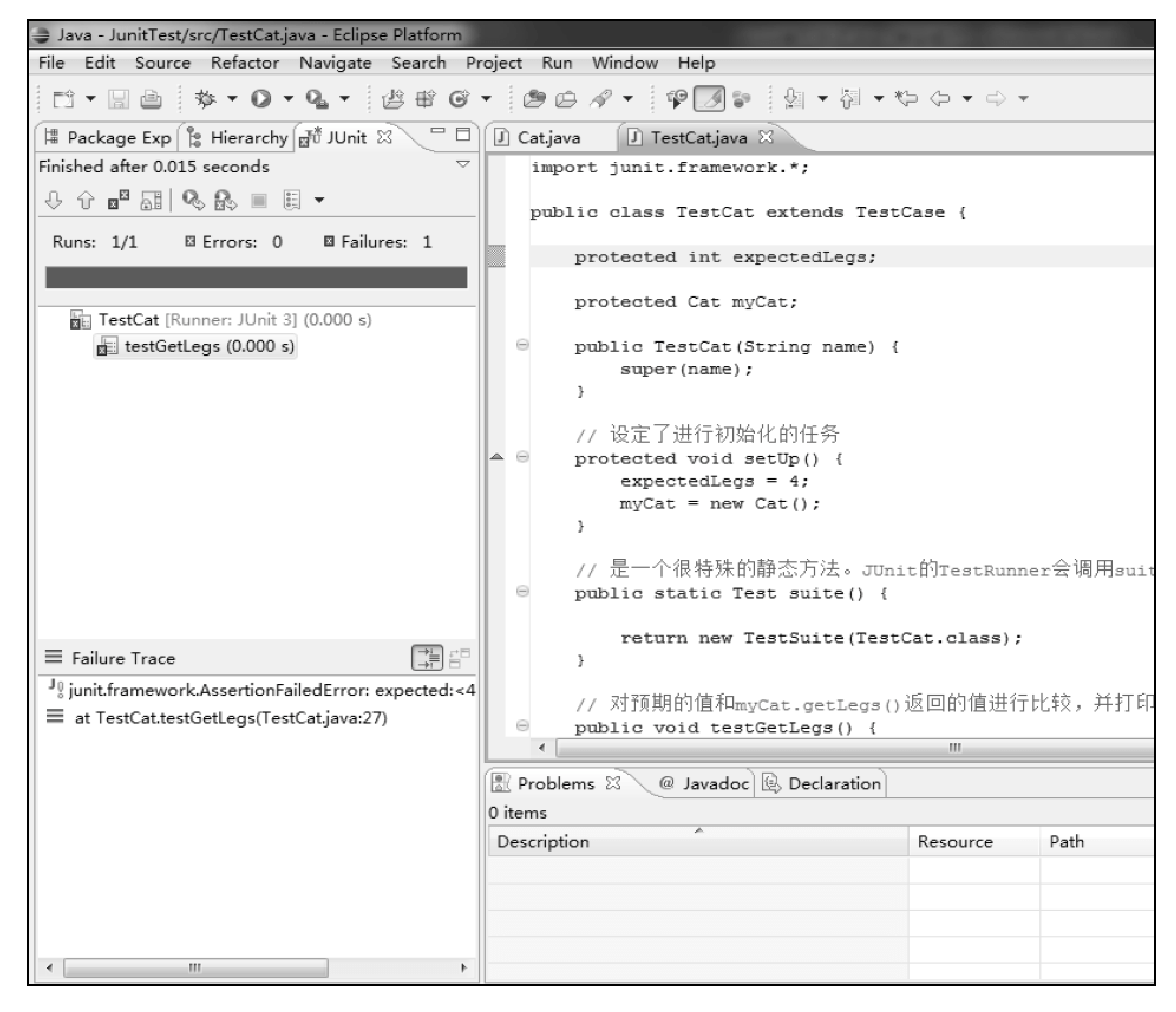


图 5-20　未通过的 JUnit 测试

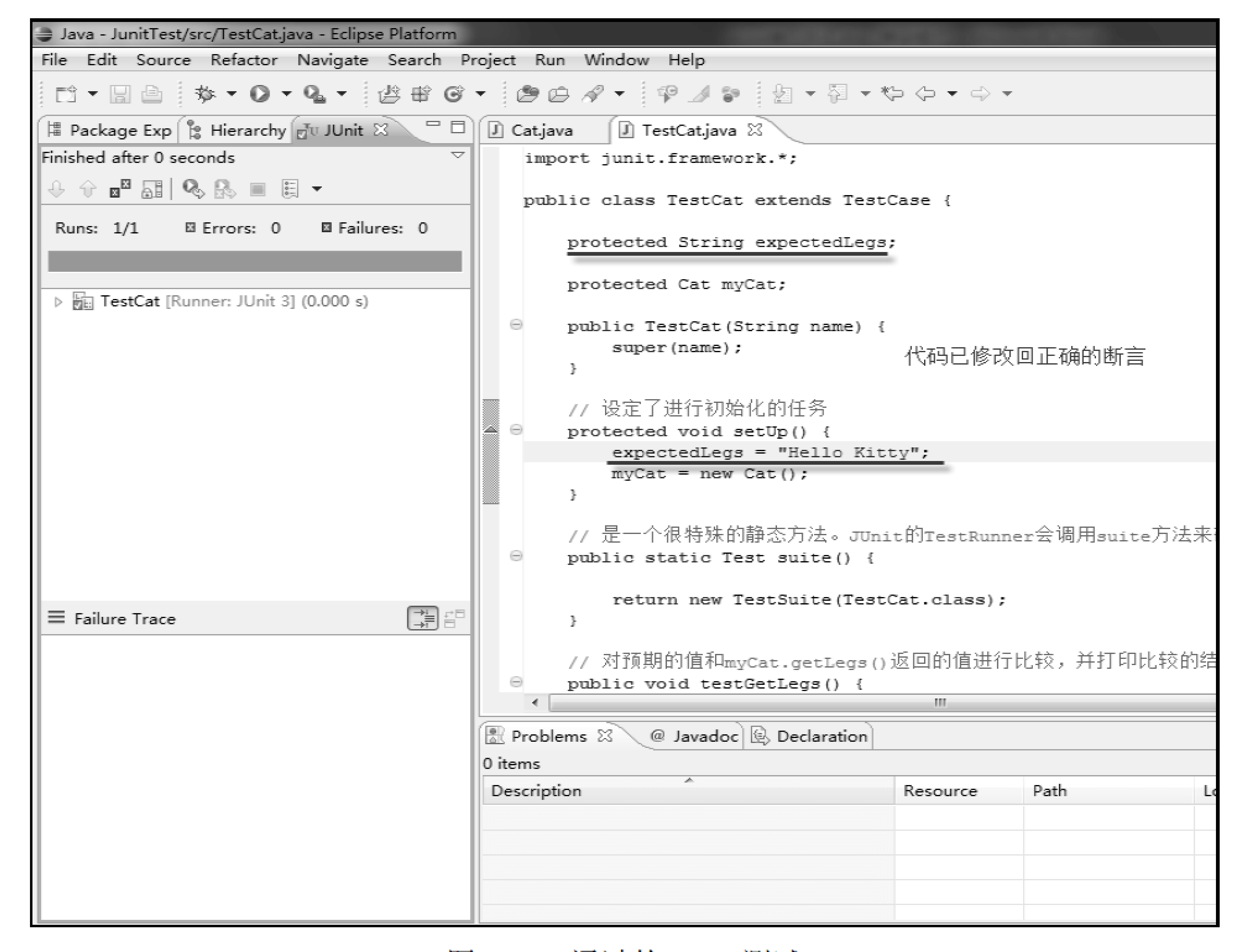


图 5-21　通过的 JUnit 测试

(2) 若售货机没有零钱找，则显示“零钱找完”的红灯亮，这时再投入 1 元硬币并按下按钮后，饮料不送出来而且 1 元硬币也退出来。

(3) 若有零钱找，则显示“零钱找完”的红灯灭，在送出饮料的同时退还 5 角硬币。

整个流程如图 5-22 所示。

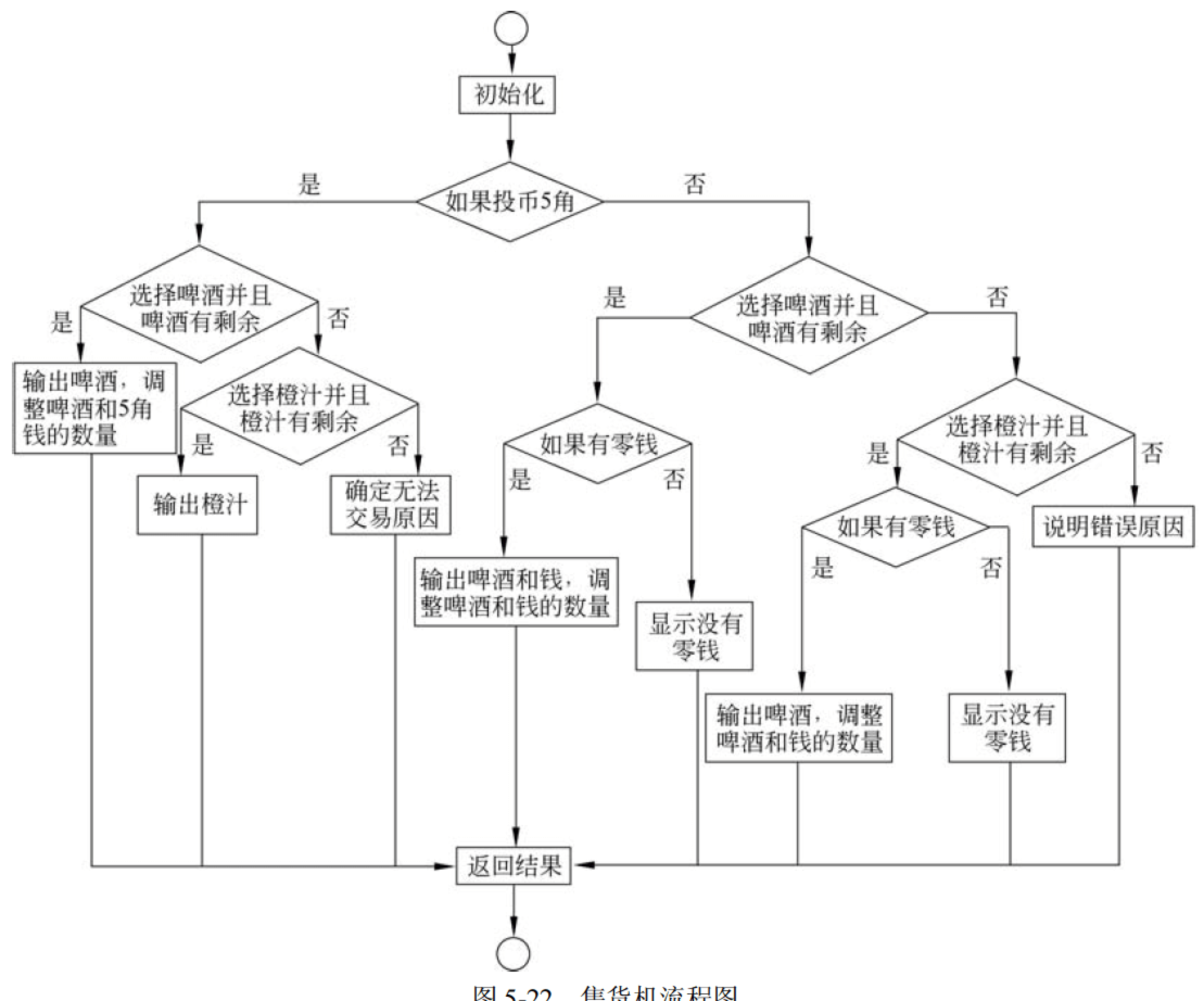


图 5-22　售货机流程图

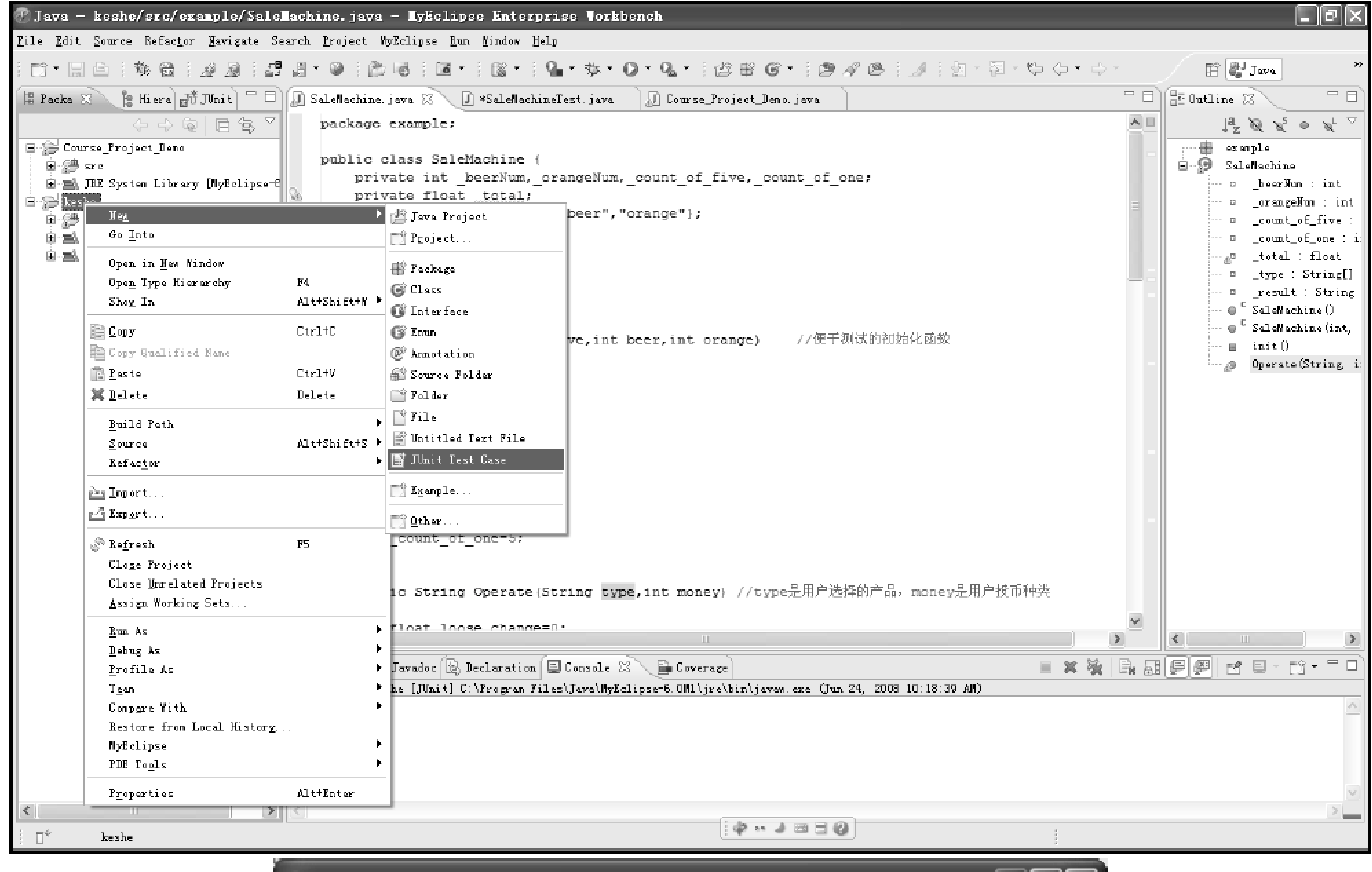
程序完整代码见光盘中的程序“自动售货机程序.C”。

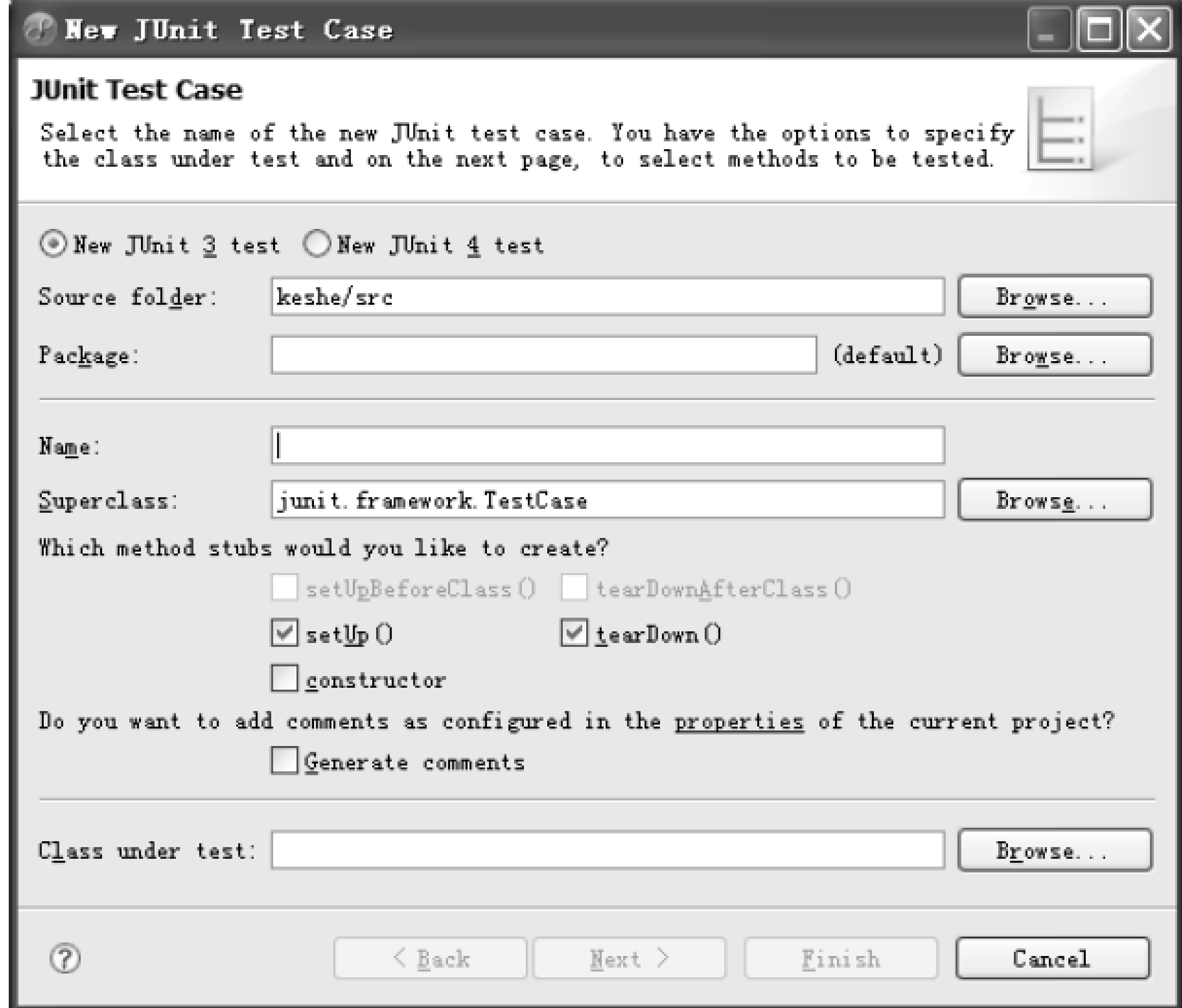
2. 测试代码

(1) 新建测试用例，选中 setUp()和 tearDown()复选框，之后出现售货机的测试用例模板，如图 5-23 所示。

我们需要按测试用例模板来编写测试用例。首先，进行测试用例的设计说明。测试用例中各字符代表含义如下：5C 代表 5 角钱，1D 代表 1 元钱，Beer 代表啤酒，OrangeJuice 代表橙汁，Coca-Cola 代表可口可乐。

(2) 根据售货机程序，这里编写了 13 个测试用例，如表 5-1～表 5-13 所示。





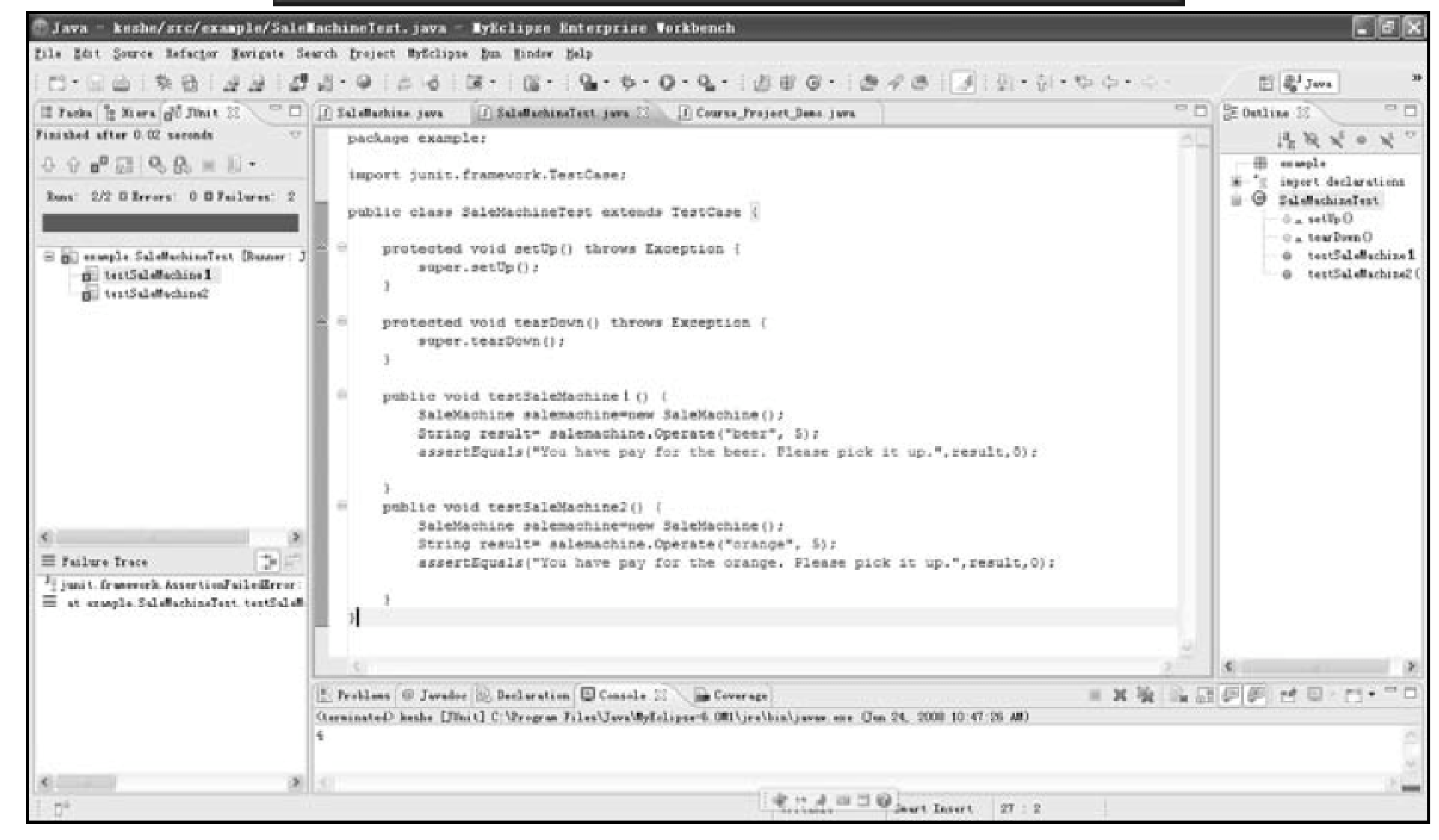


图 5-23　售货机的测试用例模板

表 5-1　售货机各资源均有剩余，用户投币 5 角，选择啤酒

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Beer	5C	各资源剩余	Input Information Type: Beer; Money: 5 Cents; Change: 0 Current State Beer: 5 Orange Juice: 6 5 Cents: 7 1 Dollar: 6	与预期相同

表 5-2　售货机各资源均有剩余，用户投币 5 角，选择橙汁

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
OrangeJuice	5C	各资源剩余	Input Information Type: OrangeJuice;Money:5 Cents;Change: 0 Current State Beer: 6 Orange Juice: 5 5 Cents: 7 1 Dollar: 6	与预期相同

表 5-3　售货机各资源均有剩余，用户投币 1 元，选择啤酒

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Beer	1D	各资源剩余	Input Information Type: Beer; Money: 1 Dollar;Change: 5 Cents Current State Beer: 5 Orange Juice: 6 5 Cents: 5 1 Dollar: 7	与预期相同

表 5-4　售货机各资源均有剩余，用户投币 1 元，选择橙汁

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
OrangeJuice	1D	各资源剩余	Input Information Type:OrangeJuice;Money:1Dollar;Change:5 Cents Current State Beer: 6 Orange Juice: 5 5 Cents: 5 1 Dollar: 7	与预期相同

表 5-5　售货机没有零钱，用户投币 1 元，选择啤酒

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Beer	1D	没有零钱	Failure Information Change Shortage	与预期相同

表 5-6　售货机没有零钱，用户投币 1 元，选择橙汁

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
OrangeJuice	1D	没有零钱	Failure Information Change Shortage	与预期相同

表 5-7　售货机没有啤酒，用户投币 1 元，选择啤酒

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Beer	1D	没有啤酒	Failure Information Beer Shortage	与预期相同

表 5-8　售货机没有橙汁，用户投币 5 角，选择橙汁

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
OrangeJuice	5C	没有橙汁	Failure Information OrangeJuice Shortage	与预期相同

表 5-9　售货机各资源均有剩余，用户投币错误，选择啤酒

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Beer	除 1D 或 5C 之外	各资源剩余	Failure Information Money Error	与预期相同

表 5-10　售货机各资源均有剩余，用户投币 1 元，选择可口可乐

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Coca-Cola	1D	各资源剩余	Failure Information Type Error	与预期相同

表 5-11　售货机没有啤酒，用户投币 5 角，选择啤酒

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Beer	5C	没有啤酒	Failure Information Beer Shortage	与预期相同

表 5-12　售货机各资源均有剩余，用户投币 5 角，选择可口可乐

输入值 type	输入值 money	状　态	预 期 输 出	实 际 情 况
Coca-Cola	5C	各资源剩余	Failure Information Type Error	与预期相同

表 5-13　售货机没有橙汁，用户投币 1 元，选择橙汁

输入值 type	输入值 money	状　　态	预 期 输 出	实 际 情 况
OrangeJuice	1D	没有橙汁	Failure Information OrangeJuice Shortage	与预期相同

测试用例的完整代码见光盘中的程序“自动售货机测试用例.java”。

3. 测试顺序

JUnit 会以如下顺序执行测试(大致的代码)：

```
try {
      CalculatorTest test = new CalculatorTest (); // 建立测试类实例
      test.setUp(); // 初始化测试环境
      test.testAdd(); // 测试某个方法
      test.tearDown(); // 清理资源
}catch…
```

setUp()用于建立测试环境，这里创建一个 SaleMachine 类的实例；tearDown()用于清理资源，如释放打开的文件等。以 test 开头的方法被认为是测试方法，JUnit 会依次执行 test×××()方法。在 testAdd()方法中，对 add()的测试选择两个数——10 和 50。如果方法返回值与期待结果相同，则 assertEquals 不会产生异常。

如果有多个 test×××方法，JUnit 将会创建多个×××Test 实例。每次运行一个 test×××方法时，setUp()和 tearDown()便会在 test×××前后被调用。因此，不要在一个 testA()中依赖 testB()。

4. 测试结果

直接运行 Run|Run As|JUnit Test，就可以看到 JUnit 测试结果，如图 5-24 所示。

绿色表示测试通过，只要有一个测试未通过，就会显示红色并列出未通过测试的方法。可以试图改变 Beer 输入数据，然后再运行 JUnit 就会报告错误。

下面简单总结一下上边用到的静态类 junit.framework.Assert 的使用方法，灵活地运用这些方法对用好 JUnit 进行测试是非常有帮助的。

(1) assertEquals()方法：用来查看对象中存储的值是否是期待的值，与用于字符串比较的 equals()方法类似。

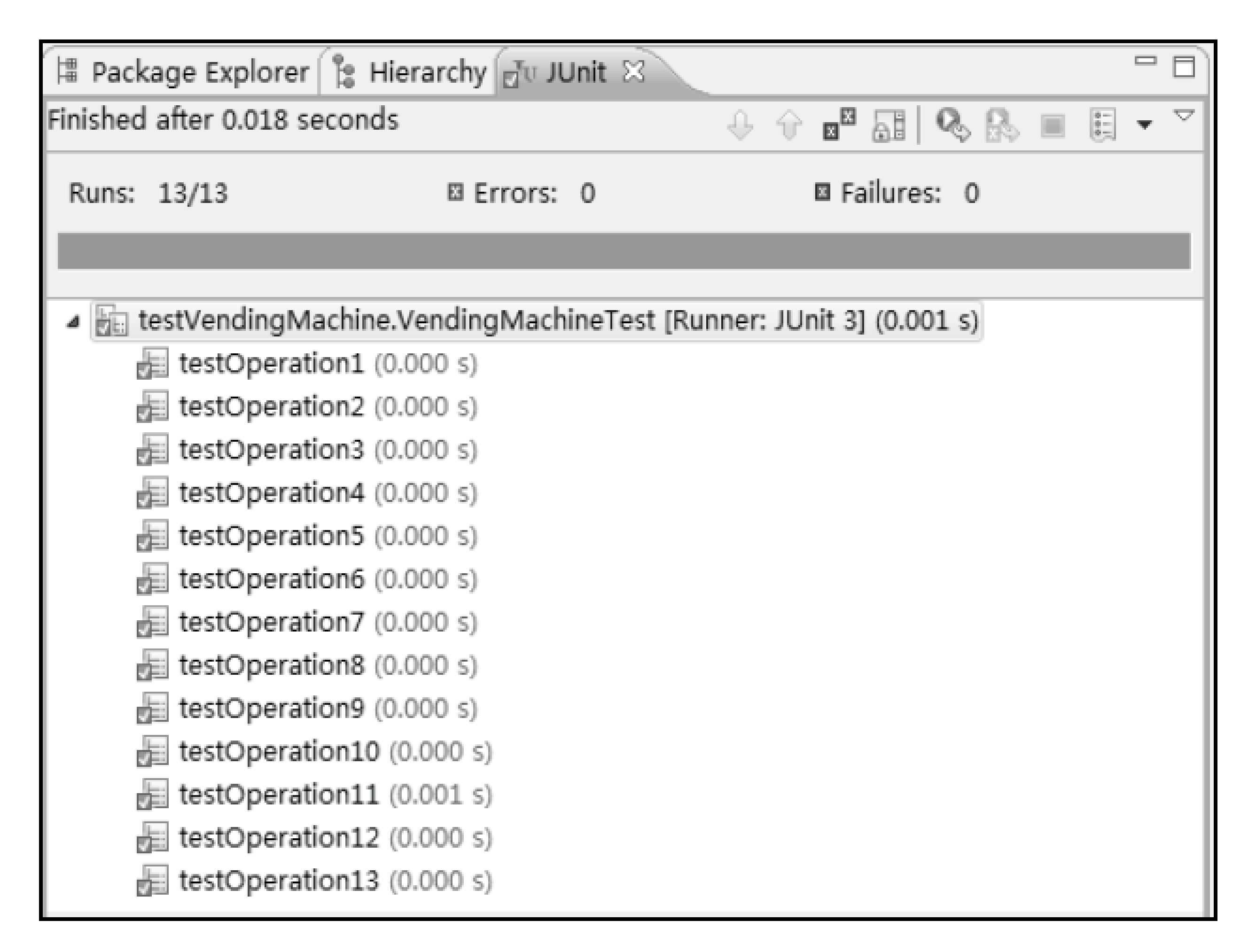
(2) assertFalse()和 assertTrue()方法：用来查看变量是否为 false 或 true。如果 assertFalse()查看的变量的值是 false，则测试成功；如果是 true，则失败。assertTrue()与之相反。

(3) assertSame()和 assertNotSame()方法：用来比较两个对象的引用是否相等和不相等，类似于通过“==”和“!=”比较两个对象。

(4) assertNull()和 assertNotNull()方法：用来查看对象是否为空和不为空。

(5) fail()方法：意为失败，用来引发错误。该方法有两个用途：一是在测试驱动开发中，由于测试用例都是在被测试的类之前编写，而写成时又不清楚其正确与否，此时就可以使用

fail()方法引发错误进行模拟；二是引发意外的错误，比如测试的内容是从数据库中读取的数据要测试其是否正确，而导致错误的原因却是数据库连接失败。



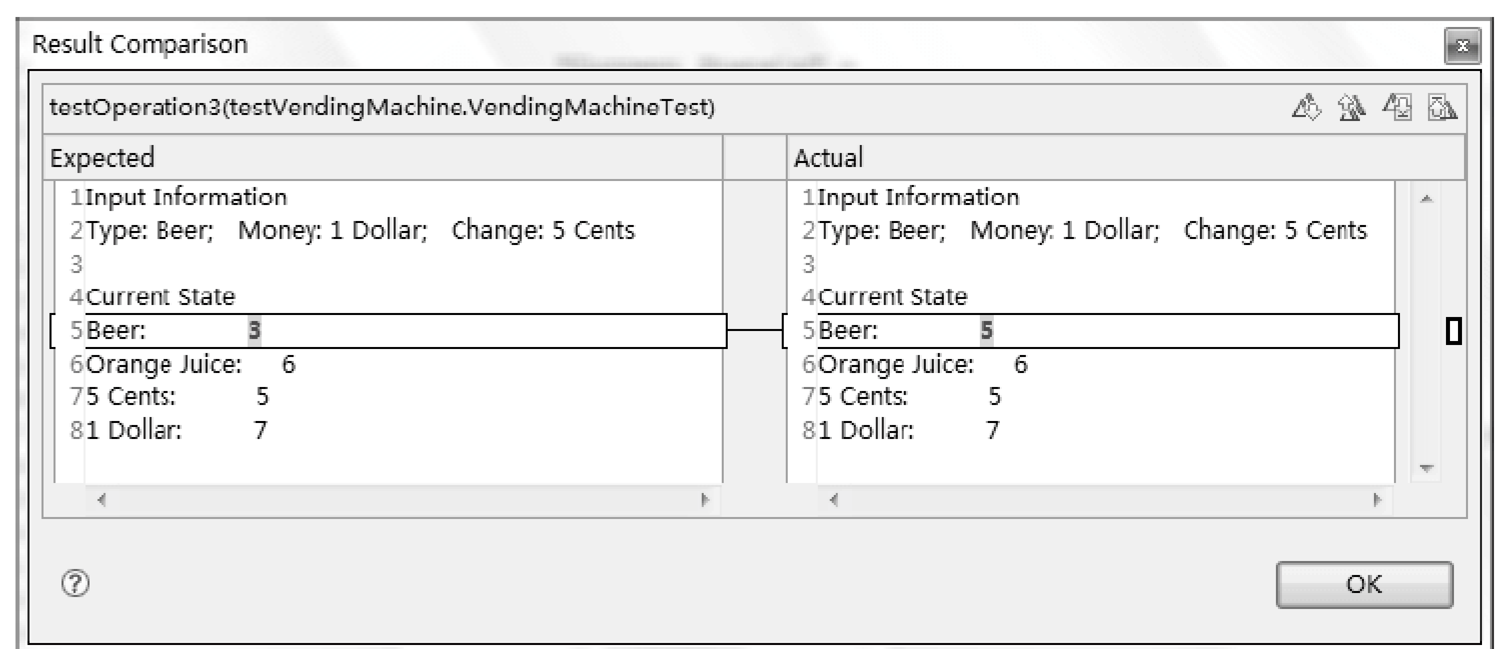


图 5-24　售货机测试无错或有错的界面显示

5.2.4　JUnit4 与 JUnit3 的区别

JUnit4 相比以前的版本而言，利用了 JDK5 的注解，使单元测试用例编写更简洁。

1. 测试用例

JUnit3 中测试用例类必须继承 TestCase 类，并且每个方法名必须以 test 开头。比如 testMethod1()，JUnit 会根据反射机制来运行测试方法。而在 JUnit4 中不必继承 TestCase，类名可以任意给定，测试方法名也可以任意命名。

在 JUnit4 中只要在测试的方法上加上注解@Test 即可，从而不必再遵循以前的一些显式约定和反射定位测试。

在 JUnit4 中如果继承了 TestCase，注解就不起作用了。并且有很重要的一点就是在 JUnit4 中继承了 TestCase 后，在 OutLine 视图中测试单个方法时，结果整个类会都运行。

2. 用例运行器

JUnit 框架靠运行器来运行测试用例。在 JUnit 中有很多个运行器(Runner)，它们负责

调用测试代码，每一个 Runner 都有各自的特殊功能，要根据需要选择不同的 Runner 来运行测试代码。

JUnit4 默认的运行器是 BlockJUnit4ClassRunner，它顺序地、一个一个地执行测试用例，JUnit4 中没有提供实现多线程执行的运行器。

3. Fixture

Fixture 是测试用例运行过程中必须执行的一组代码，它的存在主要是为测试用例或每个测试方法准备资源和销毁资源。例如数据库连接，或磁盘文件访问。

JUnit4 有两种 Fixture，一种是固定在每个测试方法前后都会执行的代码；另一种是，运行测试用例类前后固定执行的代码。

在 JUnit3 中，测试用例需要实现 setUp 和 tearDown 方法，目的就是实现上面所说的第一种 Fixture，而在 JUnit4 中无须这样，可以自定义需要在测试前和测试后的方法名，只要在方法前加上@before、@after 注解就可以了。

对于第二种 Fixture，JUnit4 引入了类范围的 setUp()和 tearDown()方法。任何用@BeforeClass 注释的方法都将在该类中的测试方法运行之前运行一次，而任何用@AfterClass 注释的方法都将在该类中的所有测试都运行之后运行一次。

4. 异常测试

异常测试是 JUnit4 中的最大改进。JUnit3 的异常测试是在引发异常的代码中放入 try 块，然后在 try 块的末尾加入一个 fail()语句。

在 JUnit4 中，可以编写引发异常的代码，并使用注释来声明该异常是预期的：如果没有异常引发或者引发一个不同的异常，那么测试就将失败。@Test(expected=ArithmeticException.class)注解表明测试方法会预期引发指定类型的异常。

5. 参数化测试

为了简化类似的测试，JUnit4 提出了“参数化测试”的概念，只写一个测试函数，把这若干种情况作为参数传递进去，一次性完成测试。

JUnit4 中参数化测试要点如下。

(1) 测试类必须由 Parameterized 测试运行器修饰。

(2) 准备数据。数据的准备需要在一个方法中进行，该方法需要满足一定的要求：①该方法必须由 Parameters 注解修饰；②该方法必须为 public static 的；③该方法必须返回 Collection 类型；④该方法的名字不做要求；⑤该方法没有参数。

6. 常用断言

1) 添加了新断言语句

JUnit4 添加了两个比较数组的 assert() 方法：

```
public static void assertEquals(Object[] expected, Object[] actual)
public static void assertEquals(String message, Object[] expected, Object[] actual)
```

这两个方法以最直接的方式比较数组：如果数组长度相同，且每个对应的元素相同，则两个数组相等，否则不相等。

2) 提供了新的断言语句

JUnit4 结合 Hamcrest 提供了新的断言语句 assertThat。

assertThat 的基本语法如下：

```
assertThat(T actual, Matcher matcher)
assertThat(String reason, T actual, Matcher matcher)
```

actual 是接下来想要验证的值；matcher 是使用 Hamcrest 匹配符来表达的对前面变量所期望的值的声明，如果 actual 值与 matcher 所表达的期望值相符，则断言成功，否则断言失败。reason 是自定义的断言失败时显示的信息。

例如，如果测试的字符串 testedString 包含子字符串“junit4”则断言成功：

```
assertThat(testedString, containsString( "junit4" ))
```

5.3　CppUnit 单元测试工具

CppUnit 也是 xUnit 家族中的一员，它是一个用 C++语言实现的单元测试框架。它的第一个版本是 Michael Feathers 由 JUnit 移植而来的，目前的版本为 1.12.0，源代码可通过网址http://sourceforge.net/projects/cppunit下载得到。该框架目前受到 GNU LGPL(Lesser General Public License)的保护。

CppUnit 按照层次来管理测试，最底层的就是测试用例(TestCase)。当有了几个 TestCase 以后，可以把它们组织成 Test Fixture。在 Test Fixture 中，可以建立被测试的类的实例，并编写 TestCase 对类实例进行测试。当有了多个 TestFixture 后，就可以使用 TestSuite 来对测试进行管理。可通过设计 TestSuite 中多个 TestCase 的测试内容和调用顺序来测试一组相关代码的正确性。而这一个或一组测试用例的测试对象被称为 TestFixture。

通常可以通过派生 TestFixture 类来设计某个类或某组相关功能的单元测试。此 Fixture 类定义了公共函数 setUp()来初始化每个成员变量，以及公共函数 tearDown()来释放在 setUp()中使用的资源。同时，添加一系列的测试用例(即 TestCase)，在每个测试函数中调用 CPPUNIT_ASSERT(bool)来判断某个函数或表达式的正确性。在派生类的声明中，需要通过宏 CPPUNIT_TEST 来添加对应的测试函数，并通过宏 CPPUNIT_TEST_SUITE 和 CPPUNIT_TEST_SUITE_END 来封装所有的测试函数、规定这些测试函数的执行顺序。

5.3.1　CppUnit 单元测试环境建立

1. 在 Linux 下安装

(1) 首先需要到网站 www.sourceforge.net 上下载一个 CppUnit 安装程序，如图 5-25 所示。

该下载地址不仅有 CppUnit 安装程序，还有 CppUnit 的说明文档。这里以 CppUnit 1.12.0 版进行说明（最新版本是 CppUnit 1.12.1）。

(2) 在下载了压缩包之后，就可以进行解压缩并安装了。

首先进行解压缩。在终端运行如下命令：

```
tar –xzf cppunit-1.12.0.tar.gz
```

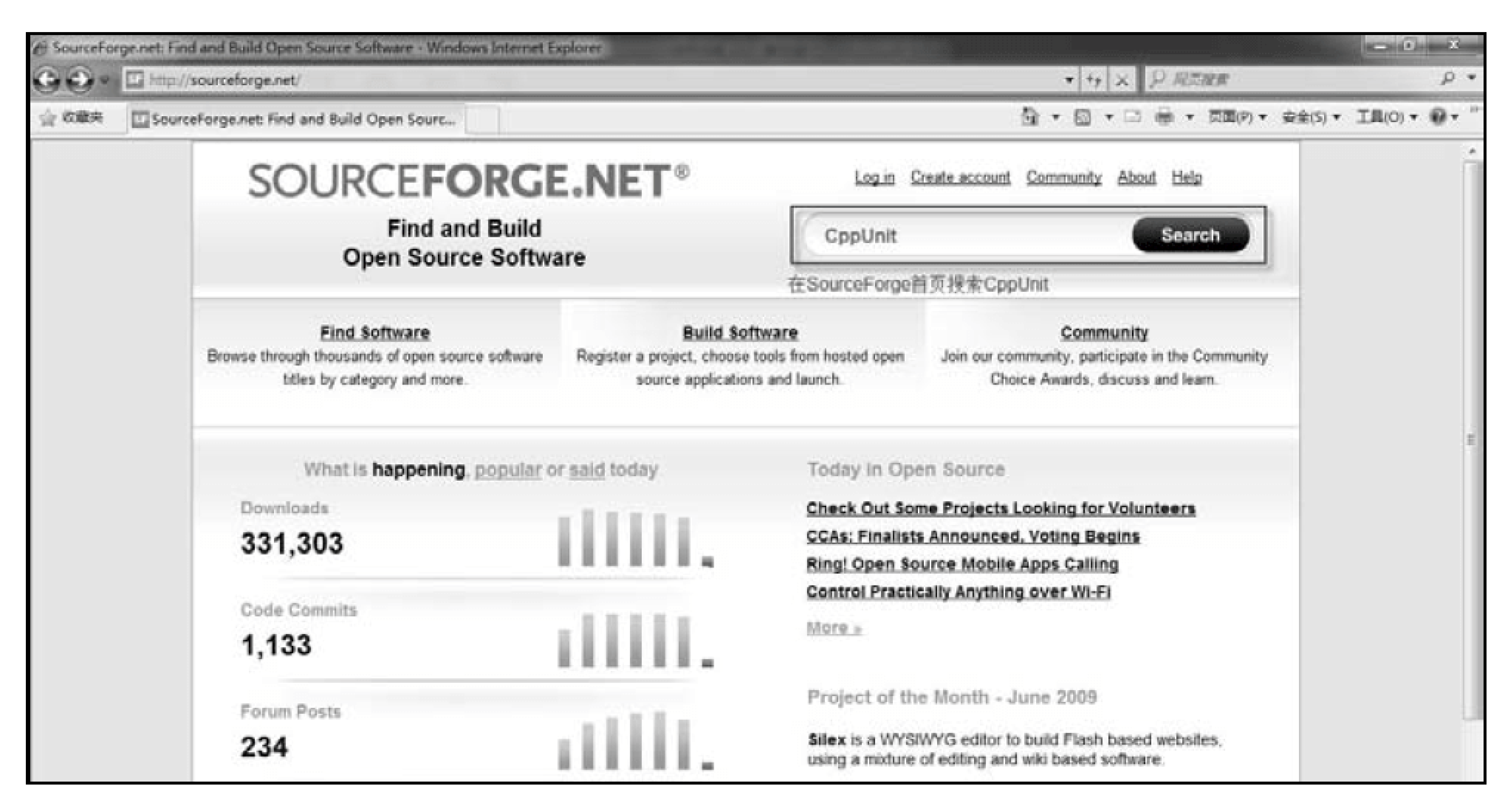


图 5-25　在 SourceForge 首页搜索 CppUnit

(3) 进入压缩包解压到的目录中：

```
cd cppunit-1.12.0
```

(4) 依次执行下面的命令，安装 CppUnit：

```
./configure
make
make check
make install
```

几乎每条命令都要执行几分钟，到此就已经完成安装了。

如果编写好程序后进行编译，编译器报告没有库文件，则需要把/usr/local/lib 下与 CppUnit 有关的几个库文件复制到/usr/lib 目录下。

2. 在 VC6/Windows 下安装

(1) 将 CppUnit1.12.0.tar.gz 解压缩到本地硬盘，如 D:\CPPUnit1.12.0。用 VC6 打开 CppUnit 1.12.0 中的项目文件 CppUnitLibraries.dsw，选择当前项目为 TestPlugInRunner，选择 Build 菜单下的 Batch Build 子菜单，将会打开一个对话框，选中所有的项目。

(2) 然后单击 Build 按钮，将生成 CppUnit 的库文件，其位置在目录 CPPUnit1.12.0\lib 下。

(3) 设置 VC 环境：选择 Tools|Options 菜单，选择 Directories TAB 选项页，在 show directories for 下拉列表框中选择 Include Files，增加路径 CPPUnit1.12.0\include，如

D:\CPPUnit1.12.0\include；同样增加 Libary Files 路径 CPPUnit1.12.0\LIB，Source Files 路径 CPPUnit1.12.0\SRC\CPPUNIT。

(4) 选择 Tools|Customize 菜单，在出现的对话框中选中 Add-ins and Macro files，单击 Browse，并选择 lib/TestRunnerDSPlugIn.dll。此时需要特别注意，关闭 VC6 后，在 Windows 的环境变量中设置 path 变量加一路径 cppunit\lib，如 D:\cppunit1.12.0\lib，否则以后的测试项目中会提示找不到动态链接库。

(5) CppUnit 的环境配置好了之后，可以通过手工来创建项目、编写测试代码，如下所示：

```
void CHostAppApp::RunUnitTests()
    {
        CppUnit::MfcUi::TestRunner runner;
        runner.addTest(CppUnit::TestFactoryRegistry::getRegistry().makeTest());
        runner.run();
    }
```

但是必须手工创建测试项目，这非常不方便。

3. 在 Eclipse/Linux 下安装

(1) 安装 Eclipse，安装 CDT，安装过程在第 3 章有详细介绍。

(2) 安装 CppUnit，安装步骤见“1.在 Linux 下安装”部分内容。在进行适配时，可添加–disable-shared 选项。

(3) 将 CppUnit 配置到 Eclipse 平台上。

将 CppUnit 配置到 Eclipse 平台实际上是通过在项目中加入引用头文件的方法来实现的。新建一个 C++项目，并对其属性进行修改，打开属性窗口，在编译用环境变量中加入“CPLUS_INCLUDE_PATH”，值设置为“/root/cppunit/include”(假定这就是 CppUnit 所在的文件夹)，如图 5-26 所示。

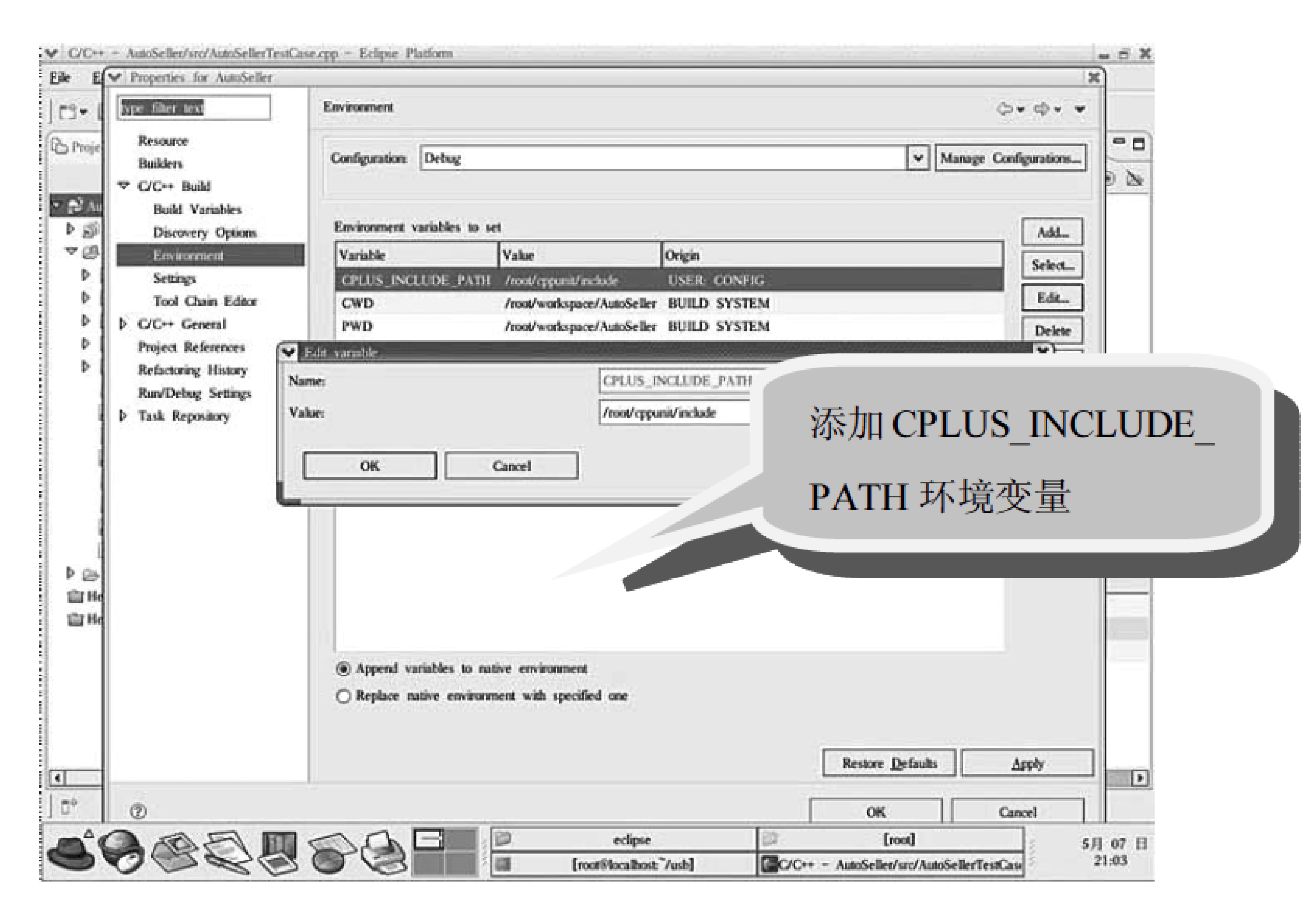


图 5-26　在 Eclipse 平台上配置 CppUnit

4. 在 MinGW/Eclipse/Windows 下安装

(1) 安装 Eclipse，安装 CDT，安装过程在第 3 章有详细介绍。

(2) 安装和配置 MinGW。

① MinGW 简介。

一个可自由使用和自由发布的 Windows 特定头文件和使用 GNU 工具集接口库的集合，支持生成本地的 Windows 程序而不需要借助第三方 C 运行时库。MinGW，即 Minimalist GNU for Windows。它是一些头文件和接口库的集合，该集合支持用户在没有第三方动态链接库的情况下使用 GCC(GNU Compiler C)来生成 Win32 程序。在基本层，MinGW 是一组包含文件和接口库，其功能是允许控制台模式的程序使用微软的标准 C 运行时库(MSVCRT.DLL)，该库在所有的 NT OS 以及所有的 Windows 95 发行版以上的 Windows OS 上有效，使用基本运行时，可以使用 GCC 编写控制台模式的符合美国标准化组织(ANSI)的程序，可以使用微软提供的 C 运行时扩展。该功能是 Win32 API 所不具备的。MinGW 的另一个组成部分是 W32 API 包，它是一组可以使用 Win32 API 的包含文件和接口库。与基本运行时相结合，就可以既使用 C 运行时(C Runtime)又使用 Win32 API。

② 安装配置。

到 MinGW 官方网站上下载 MinGW 并安装，网址为 http://www.mingw.org。

下载完成后，在安装过程中选择 download and install，当询问 Which MinGW Package do you which to install 时选择 current，然后选择所需安装的组件即可(至少需安装 MinGW base tools、G++ compiler、MinGW Make)，稍后整个编译调试环境的安装就已完成。

设置一下环境变量，如下所示：

```
MINGW_HOME = C:\MinGW(可根据自己的安装目录而定)
CLASSPATH= .;%MINGW_HOME%\lib
path= .;%MINGW_HOME%\bin
```

此时，在 cmd 中运行 GCC，若提示“gcc: no input files”，则说明环境变量设置正确，如图 5-27 所示。否则请检查其设置。

将 MinGW\bin 下的 mingw32-make.exe 在同一目录下复制一份并改名为 make.exe。

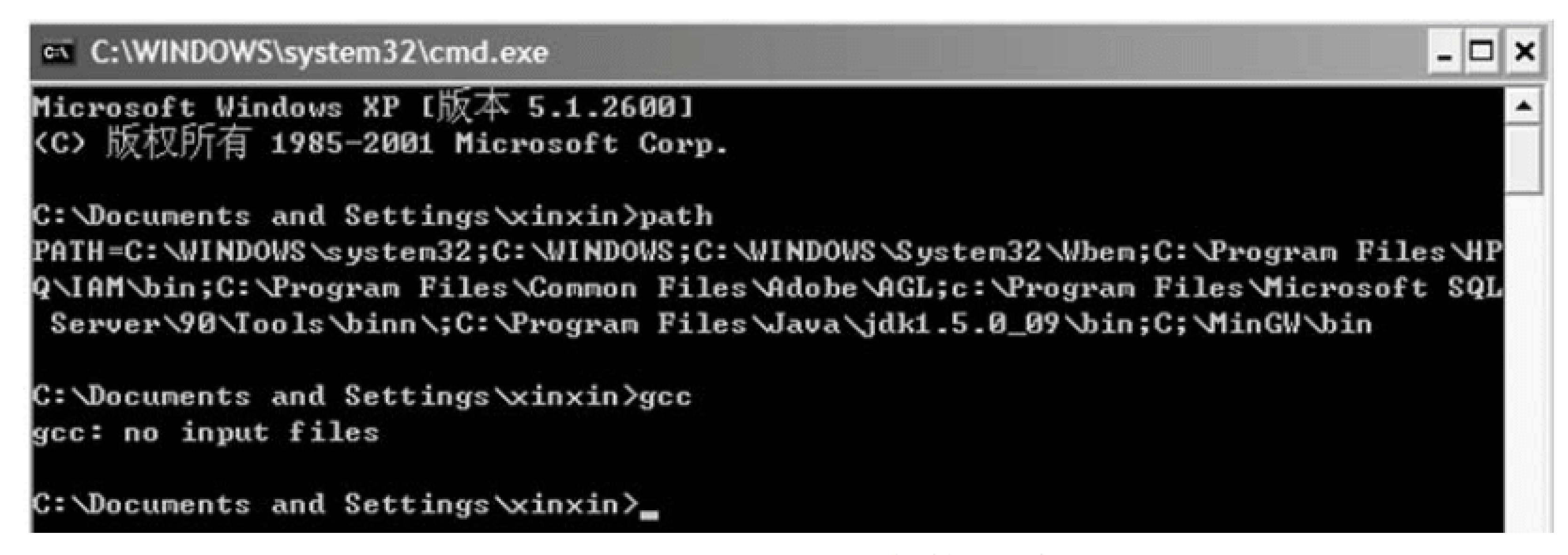


图 5-27　MinGW 安装成功

(3) 安装和配置 Cygwin。

① Cygwin 简介。

Cygwin 是一个在 Windows 平台上运行的 UNIX 模拟环境，是 Cygnus Solutions 公司开

发的自由软件(该公司开发了很多著名软件，其中 eCos 现已被 Red Hat 收购)。它对于学习 UNIX/Linux 操作环境，从 UNIX 到 Windows 的应用程序移植，以及进行某些特殊的开发工作，尤其是使用 GNU 工具集在 Windows 上进行嵌入式系统开发非常有用。随着嵌入式系统开发在国内日渐流行，越来越多的开发者对 Cygwin 产生了兴趣。Cygnus 首先对 GCC、GDB、GAS 等开发工具进行了改进，使它们能够生成并解释 Win32 的目标文件。然后，再把这些工具移植到 Windows 平台上去。一种方案是基于 Win32 API 对这些工具的源代码进行大幅修改，这样做显然需要大量工作。因此，它们采取了一种不同的方法——编写一个共享库(就是 cygwin.dll)，把 Win32 API 中没有的 UNIX 风格的调用封装在里面，也就是说，它们基于 Win32 API 编写了一个 UNIX 系统库的模拟层。这样，只要把这些工具的源代码和这个共享库连接到一起，就可以使用 UNIX 主机上的交叉编译器来生成可以在 Windows 平台上运行的工具集。以这些移植到 Windows 平台上的开发工具为基础，Cygnus 又逐步把其他的工具软件移植到 Windows 上来。这样，在 Windows 平台上运行 bash 和开发工具、用户工具时，感觉就好像在 UNIX 上工作一样。

② 配置 Cygwin。

下载 Cygwin 并安装，安装时根据提示单击“下一步”按钮即可。

安装完成后修改环境变量“path=;%Cygwin_HOME%\bin”(路径可根据自己的安装路径而定)。运行结果如图 5-28 所示。

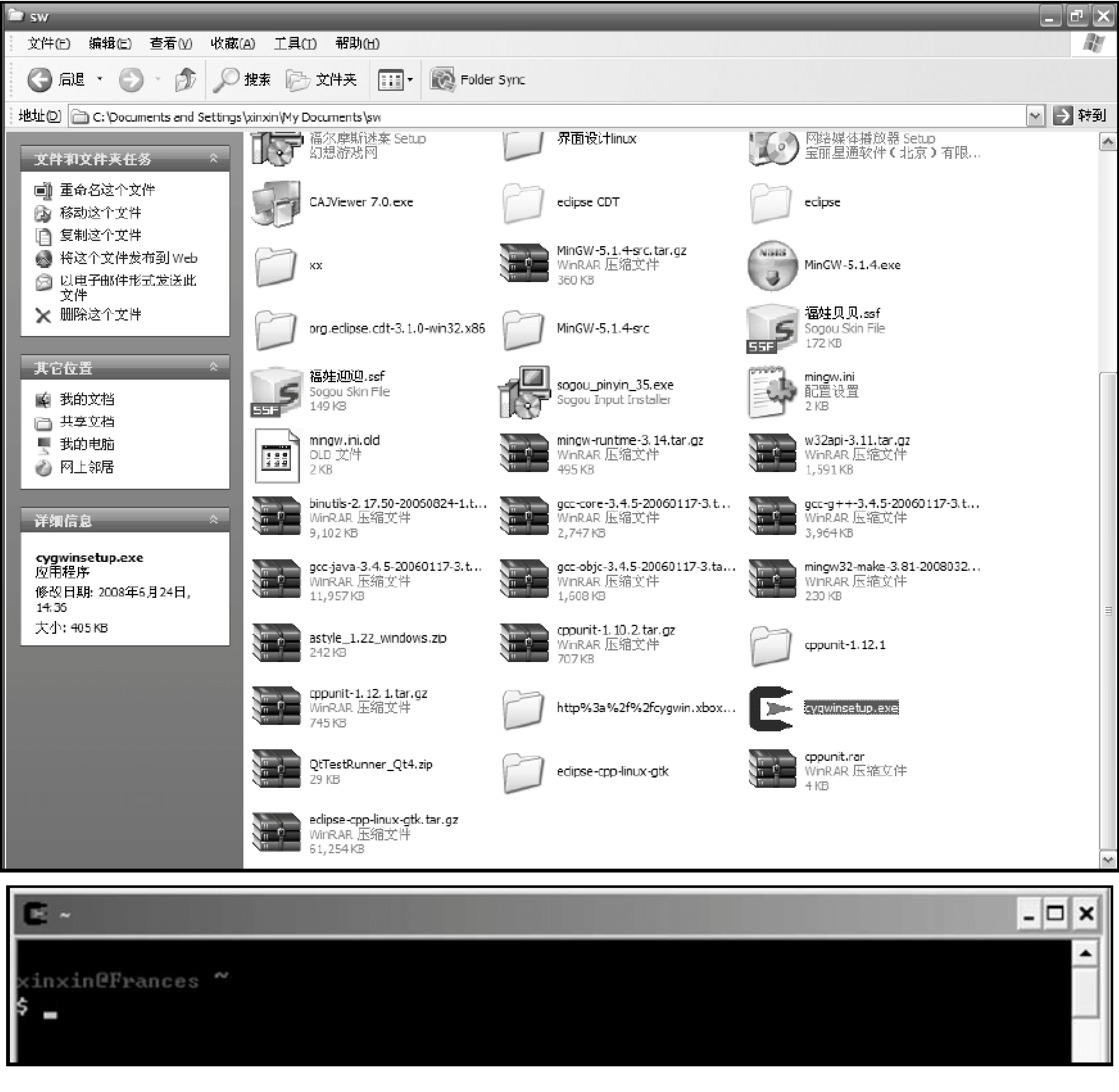


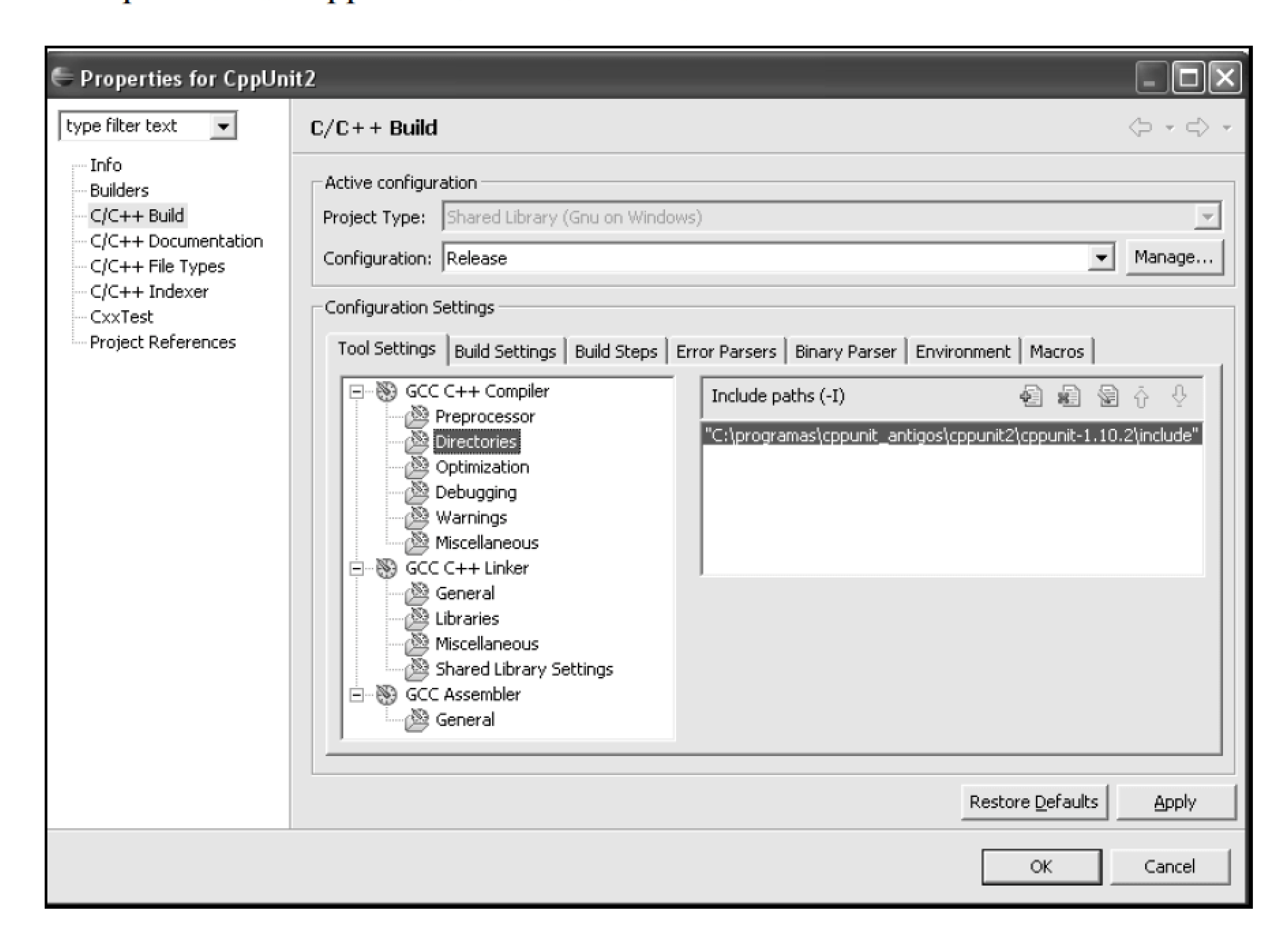
图 5-28　Cygwin 安装结果

(4) 安装和配置 CppUnit。

下载 CppUnit-1.10.2.tar.gz 并解压到指定目录(Cygwin 根目录的 home 文件夹下)，运行 msys(即刚刚配置好的 Cygwin 环境)，并在进入 CppUnit 所在目录后输入：

```
./configure
Make
Make install
```

打开 Eclipse，加载 CppUnit 插件，步骤如图 5-29 所示。



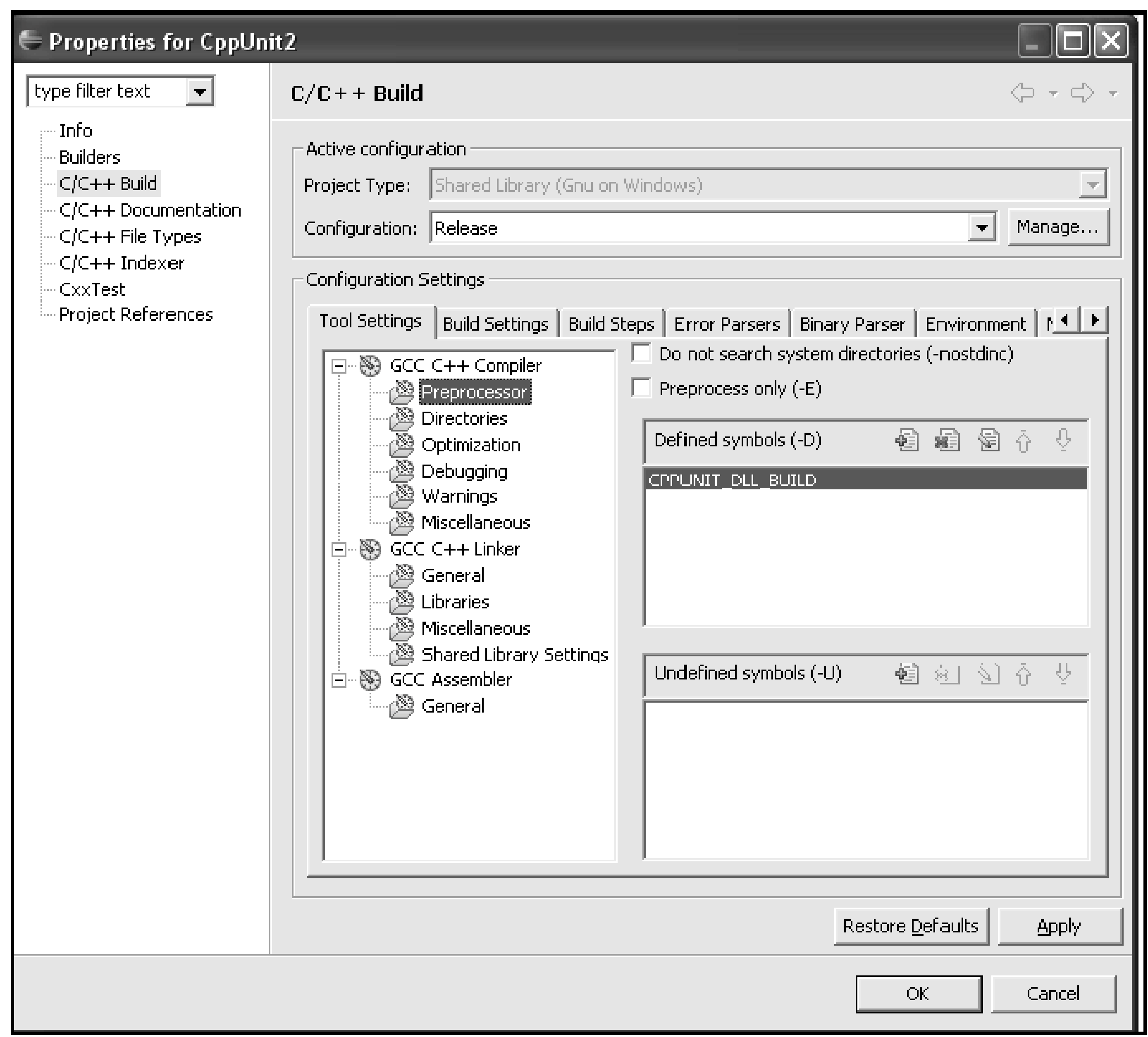


图 5-29　加载 CppUnit

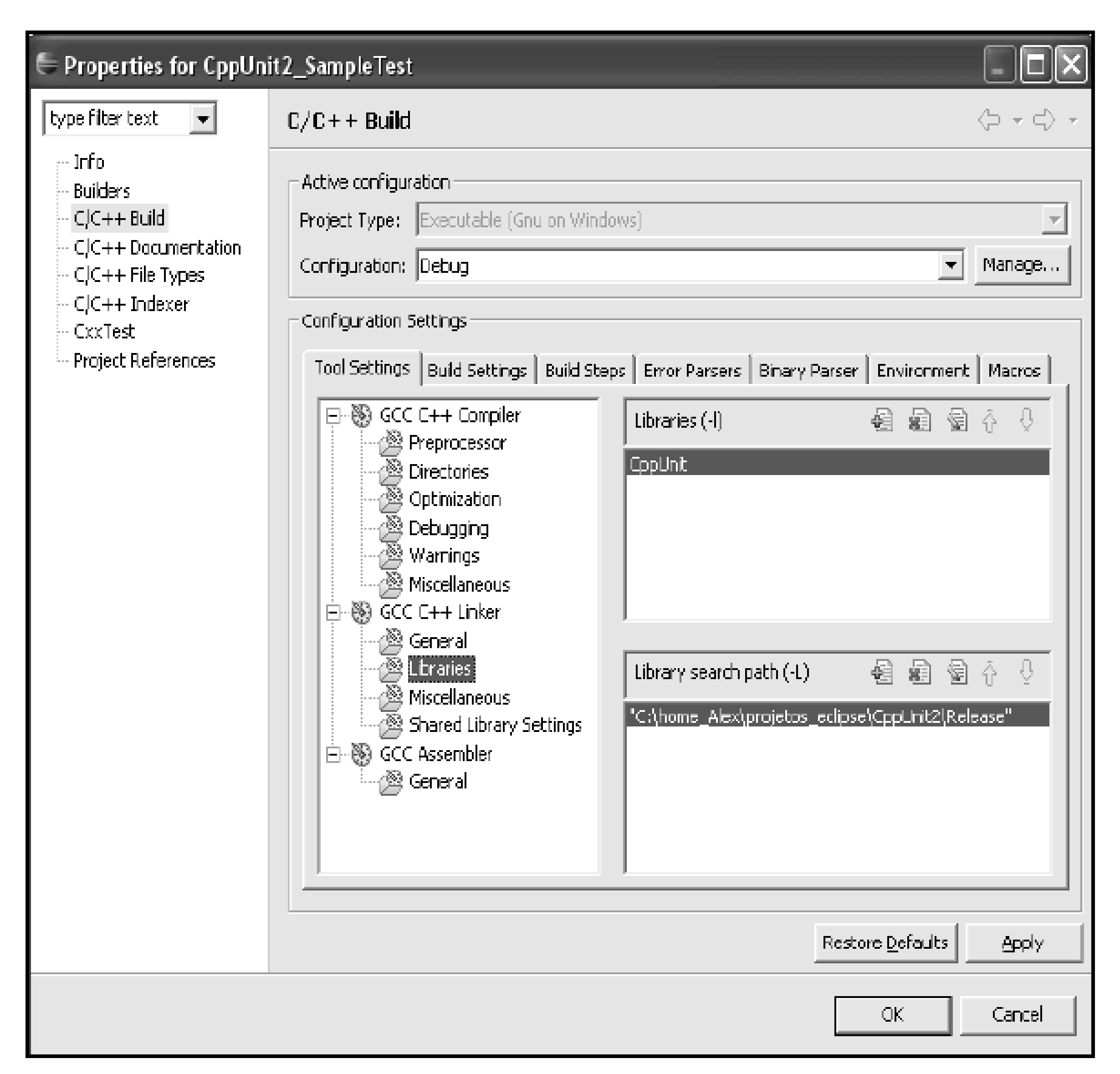


图 5-29 (续)

配置完成后，如果希望 CppUnit 模块能自动升级，可单击 help 中的软件升级。

5.3.2 CppUnit 功能和使用流程

1. CppUnit 核心部分(Core)

(1) 基本测试类：Test，TestFixture，TestCase，TestSuite。

(2) 测试结果记录：SynchronizedObject，TestListener，TestResult。

(3) 错误处理：TestFailure，SourceLine，Exception，NotEqualException。

(4) 断言：Asserter，TestAssert。

2. 输出部分(Output)

(1) 基础部件：Outputter，TestResultCollector。

(2) 衍生类：TextOutputter，CompilerOutputter，XmlOutputter。

3. 辅助部分(Helper)

TypeInfoHelper，TestFactory，TestFactoryRegistry，NamedRegistries，TestSuiteFactory，TestSuiteBuilder，TestCaller，AutoRegisterSuite，HelperMacros。

4. 扩展部分(Extension)

TestDecorator，RepeatedTest，Orthodox，TestSetUp。

5. 监听者部分(Listener)

TestSucessListener，TextTestProgressListener，TextTestResult。

6. 界面部分(UI)

TestRunner (TextUI, MfcUI, QtUI)。

7. 移植(Portability)

OStringStream。

下面通过一个例子来简单地说明 CppUnit 的使用流程。

(1) 启动 Eclipse，单击 New，新建 C++项目，在 Project name 文本框中输入项目名称，如图 5-30 所示。

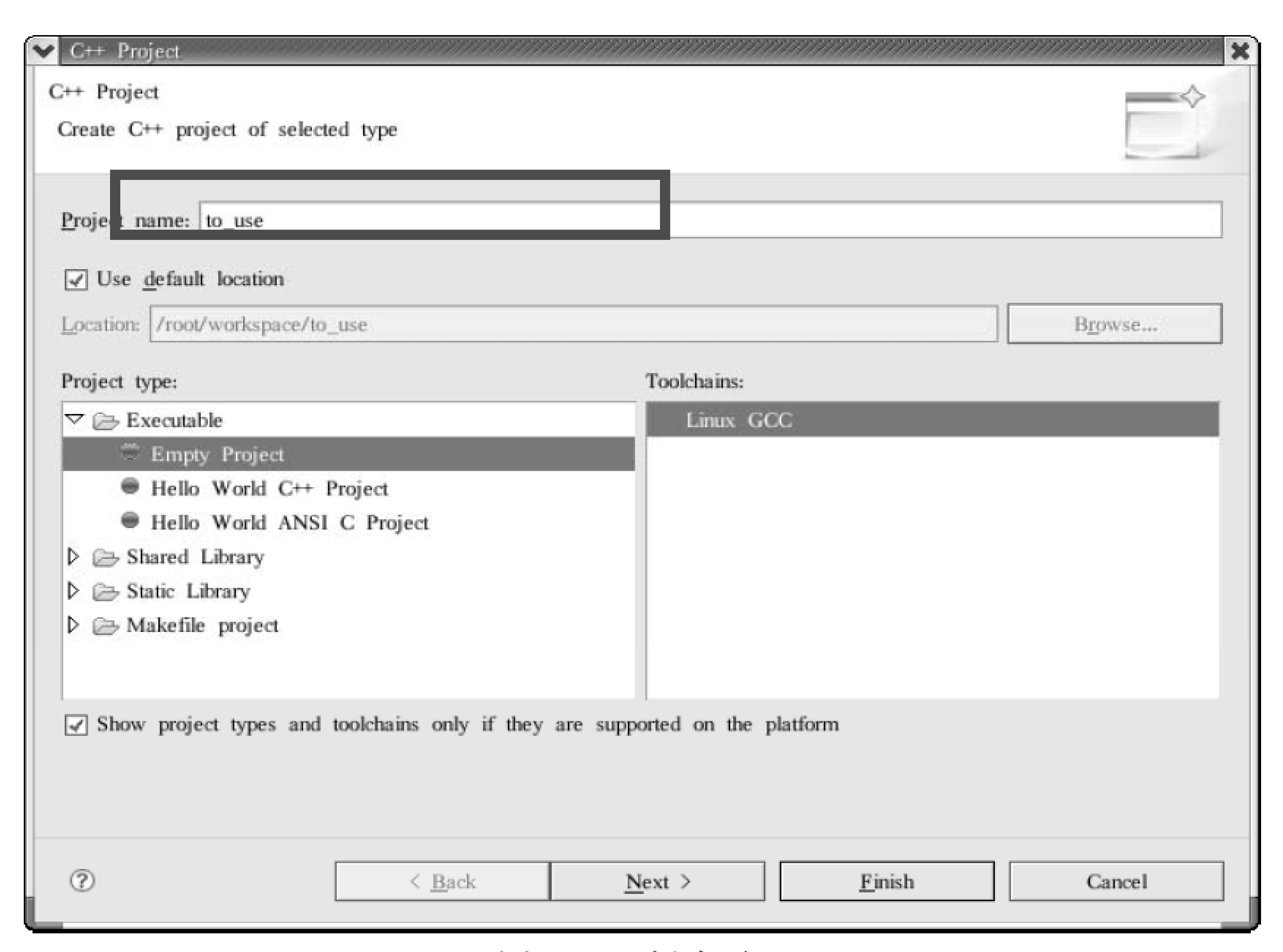


图 5-30　创建项目

(2) 在所建的 project 处右击鼠标，新建 cpp 文件，用于编写代码，如图 5-31 所示。

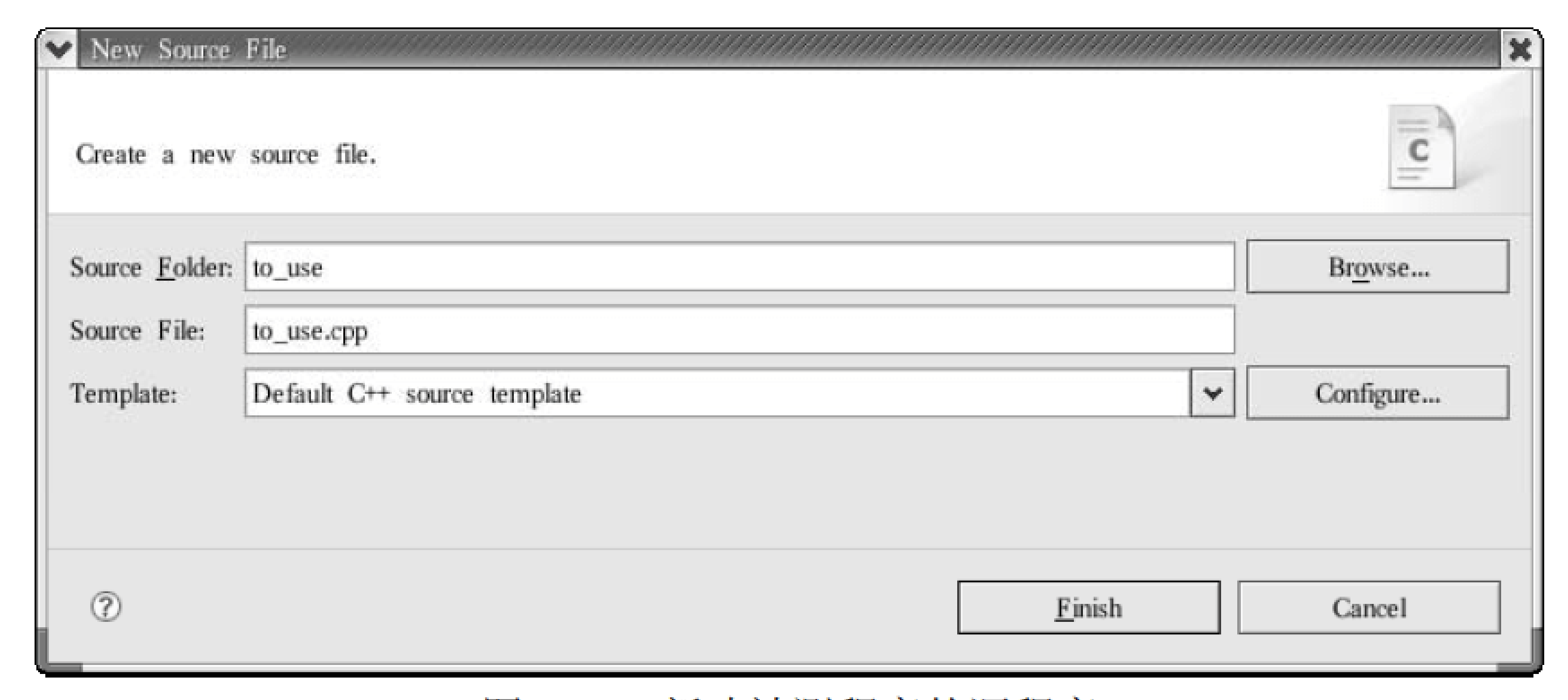


图 5-31　新建被测程序的源程序

(3) 在所建的 project 处右击鼠标，创建头文件，用于类的定义或其他声明，如图 5-32 所示。

(4) 在所建的 project 处右击鼠标，新建测试用例的头文件，并在文件中输入相关内容(注意 CppUnit 中 HelperMacros.h 文件的路径)，如图 5-33 所示。

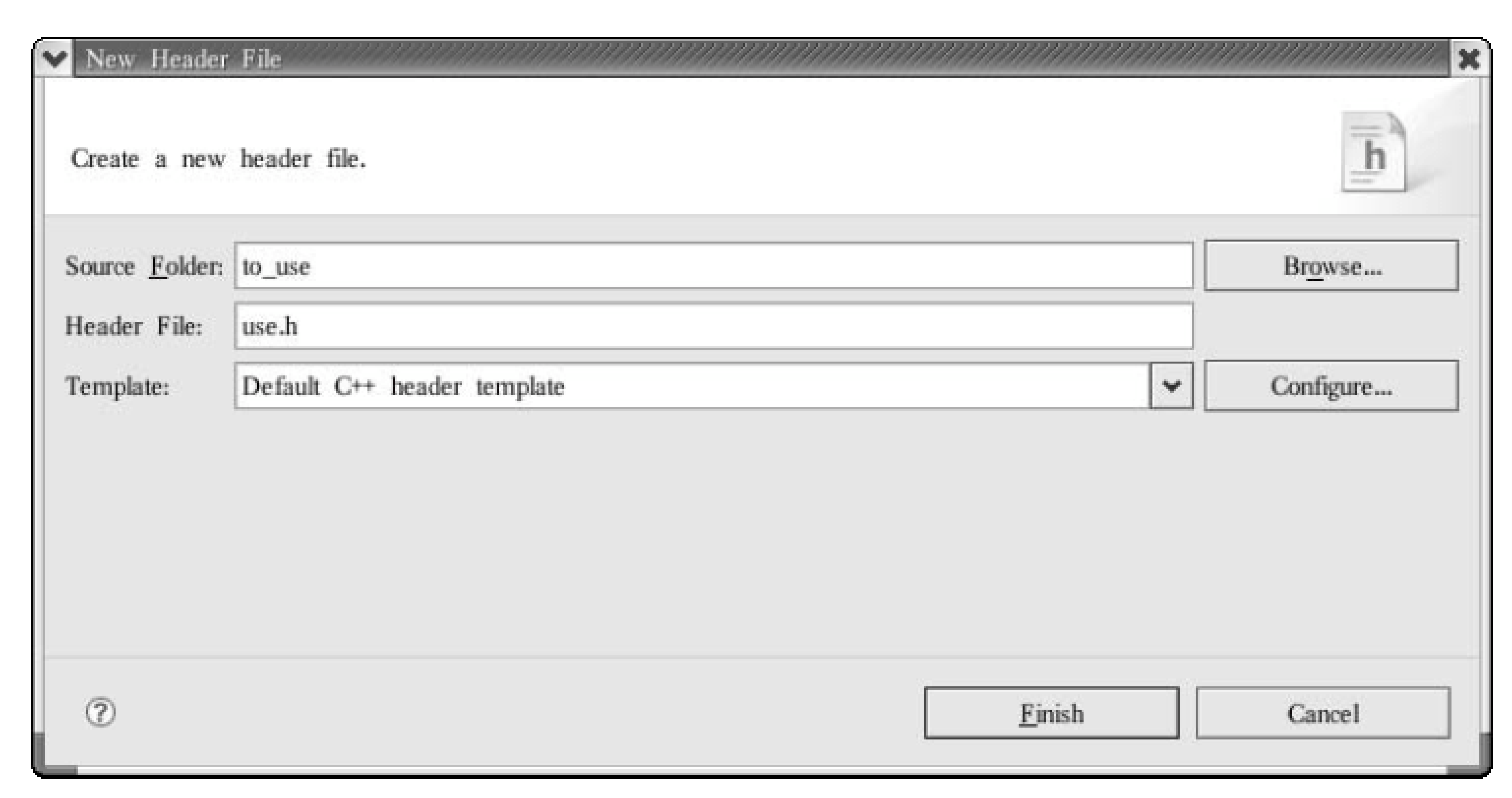
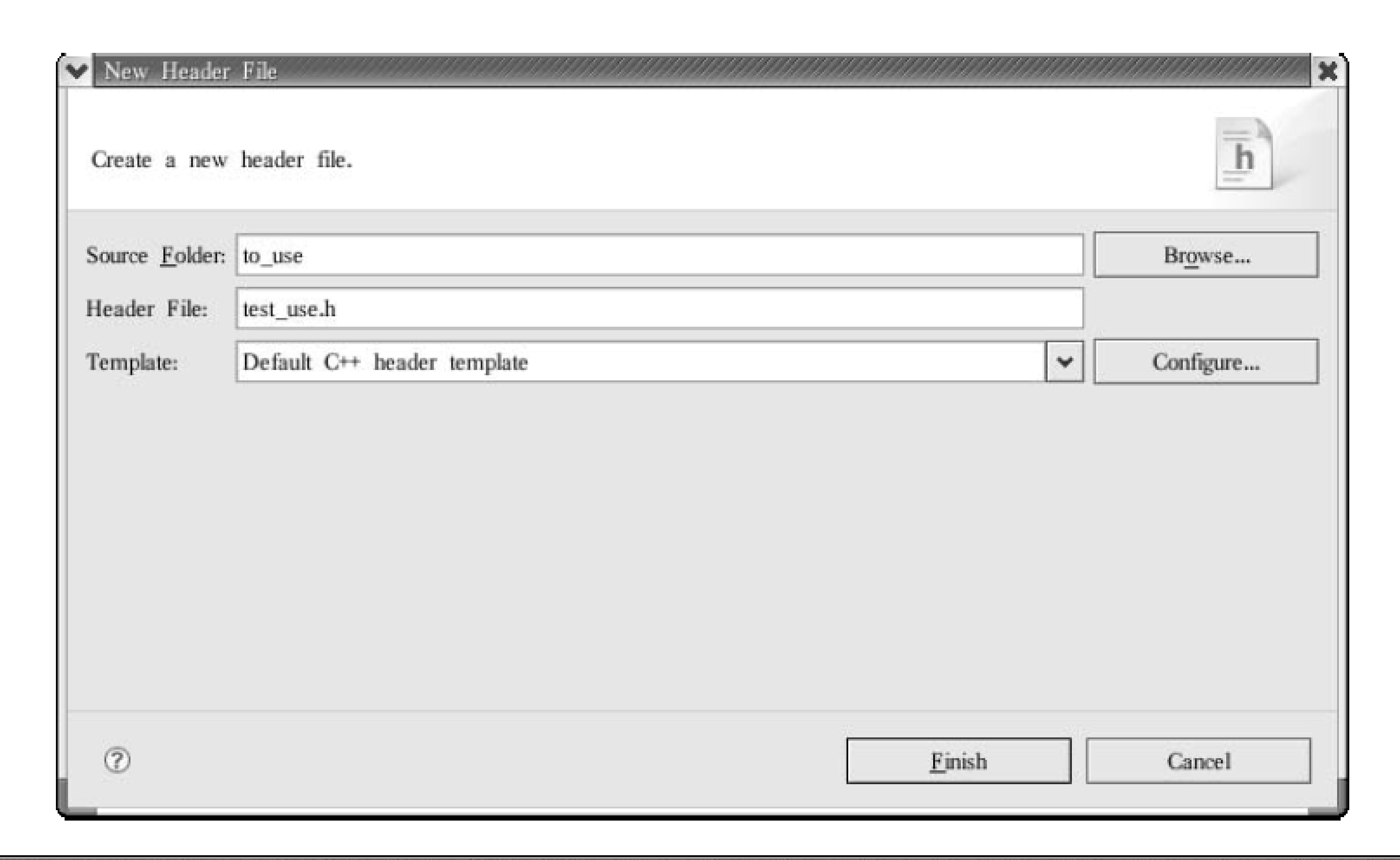


图 5-32　新建被测程序的头文件



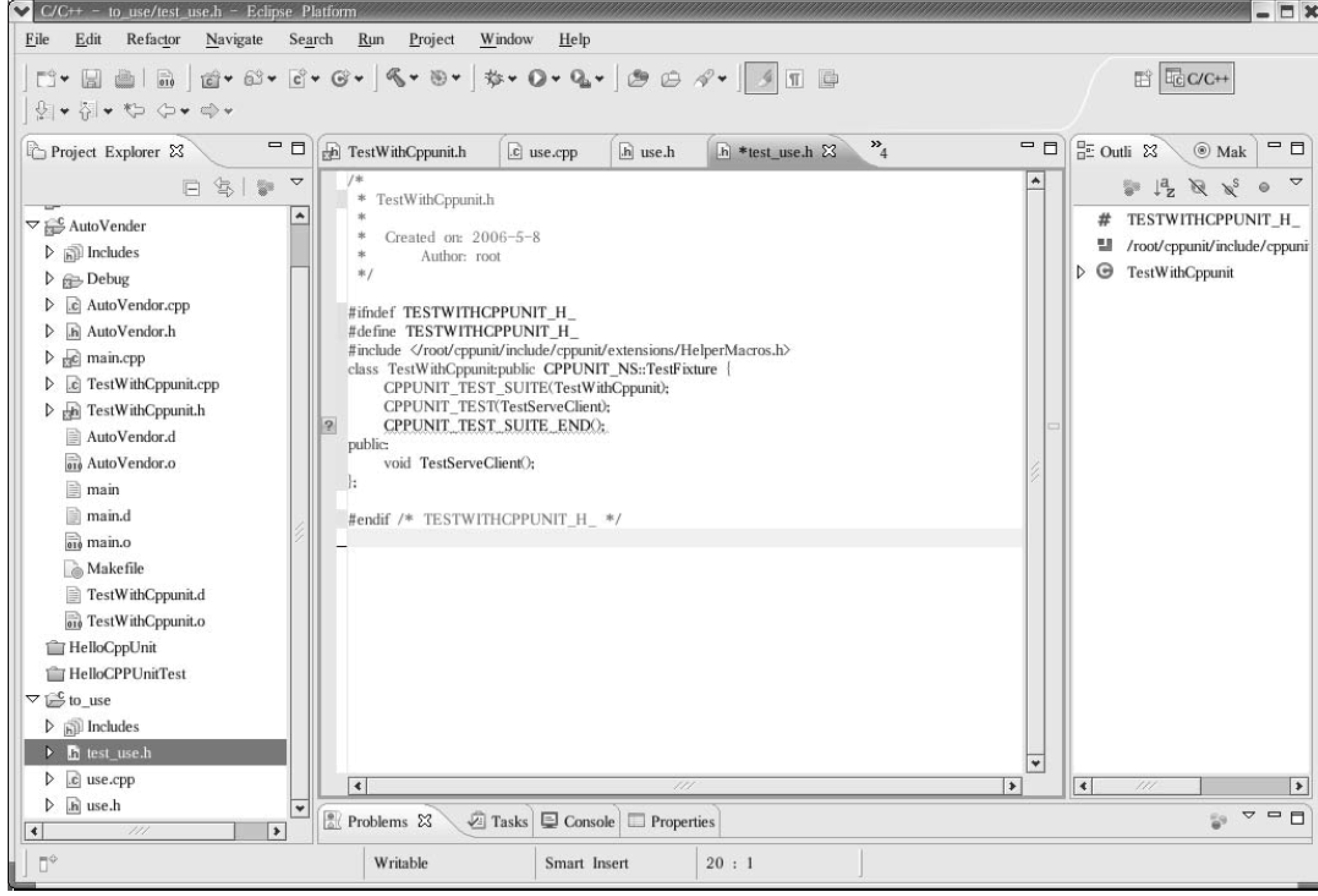
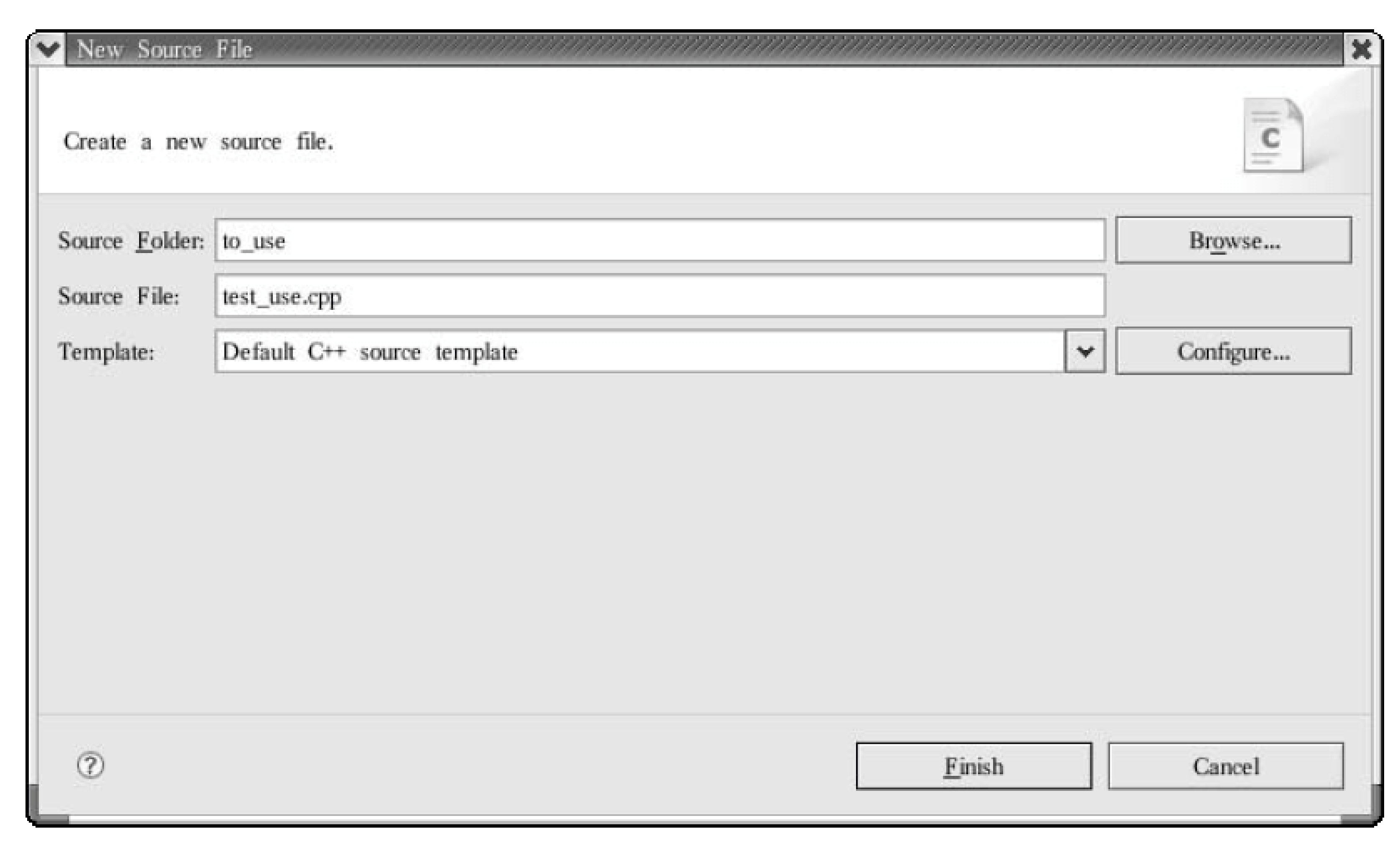


图 5-33　新建测试用例的头文件

三个宏：CPPUNIT_TEST_SUITE()，CPPUNIT_TEST()，CPPUNIT_TEST_SUITE_END()。

第一个宏声明了一个测试用例，第二个宏声明了一个测试方法，第三个宏则结束创建 TestSuite。

(5) 在所建的 project 处右击鼠标，新建测试用例的源文件(文件后缀为.cpp)，并在文件中输入测试用例代码，如图 5-34 所示。



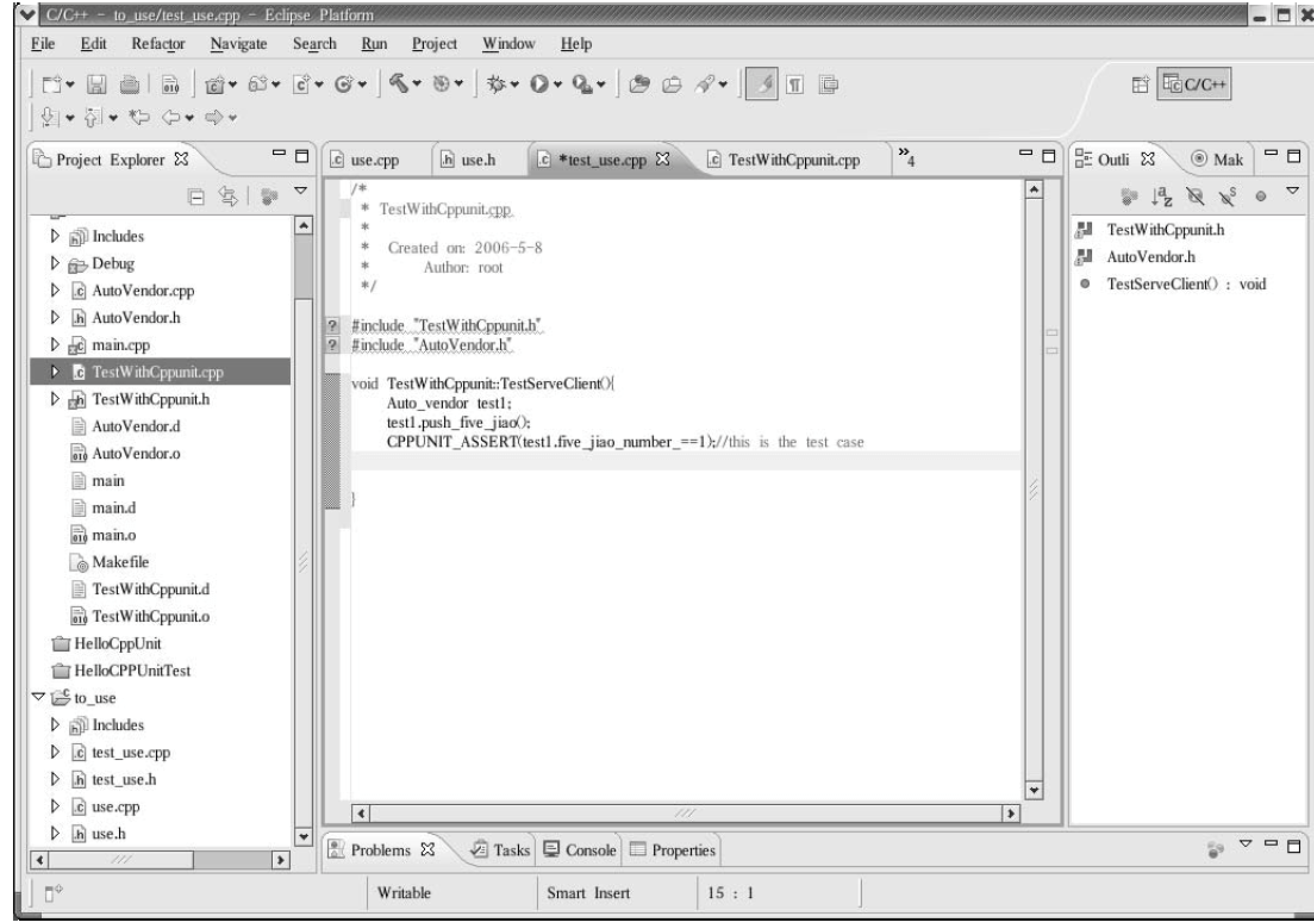
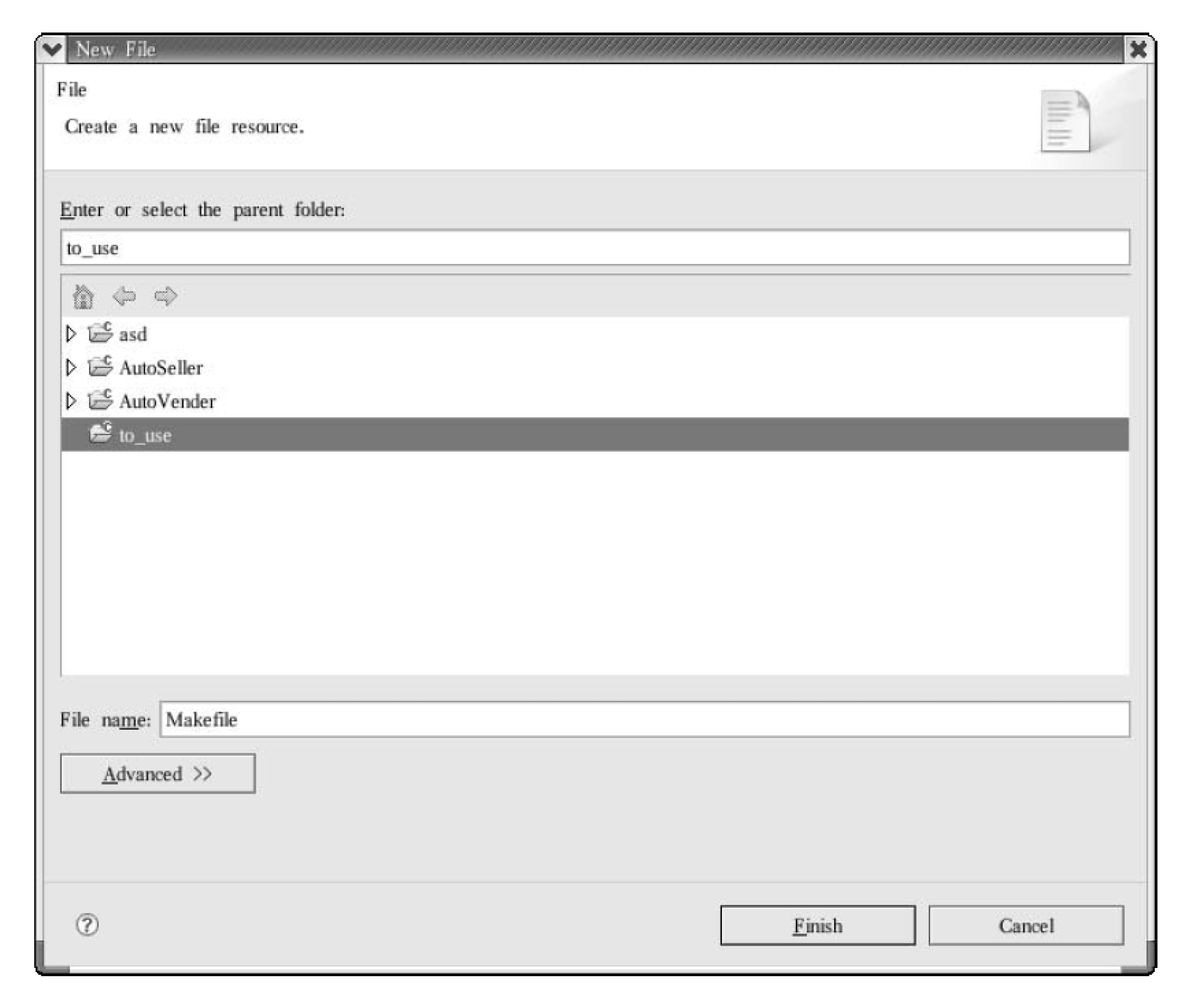


图 5-34　新建测试用例的源文件

(6) 在所建的 project 处右击鼠标，选择相应项目，创建并编写文件 Makefile(文件类型为 file，文件无后缀)，在其中指明 cppunit_lib、cppunit_config 及 object 的路径，如图 5-35 所示。



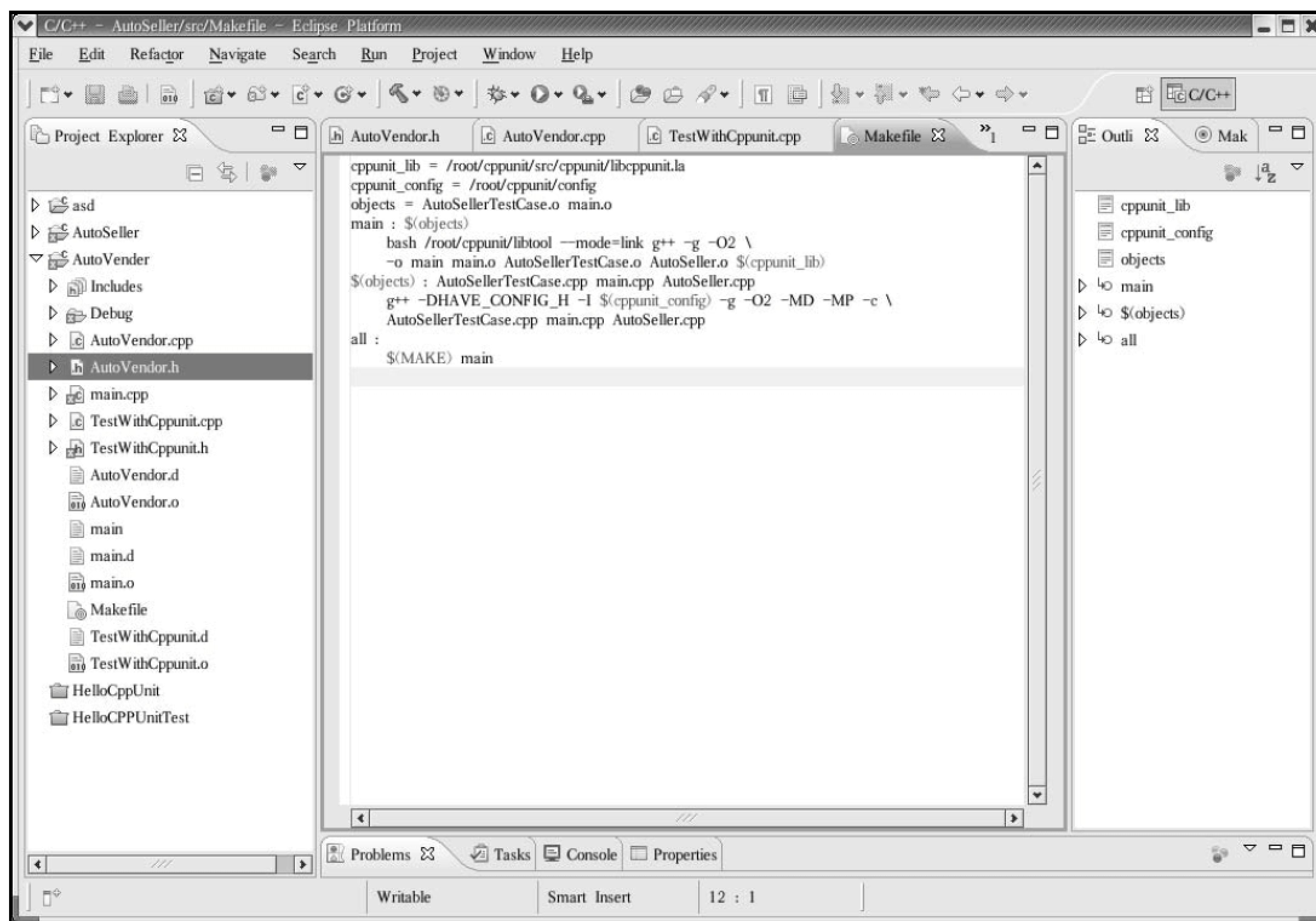


图 5-35　新建 Makefile

(7) 新建终端，进入所建项目的工作空间，对文件夹进行编译测试。执行命令：①cd workspace；②cd AutoVendor；③make；④./main。如果出现“OK”，则说明测试成功，如图 5-36 所示。

```
root@localhost:~/workspace/AutoVender
文件(F)  编辑(E)  查看(V)  终端(T)  转到(G)  帮助(H)
umount: /root/usb: not mounted
[root@localhost root]# fdisk -l

Disk /dev/hda: 120.0 GB, 120034123776 bytes
255 heads, 63 sectors/track, 14593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *      8601      9237   5116671   83  Linux
/dev/hda2          9238      9492   2048256   82  Linux swap
[root@localhost root]# vi driver.cpp
[root@localhost root]# cd jerry
[root@localhost jerry]# ls
1.png  3.png  5.png  7.png       cppunit-1.10.2  hello
2.png  4.png  6.png  AutoVendor  eclipse
[root@localhost jerry]# cd AutoVendor
[root@localhost AutoVendor]# ls
AutoVendor.cpp  AutoVendor.h  drive.cpp
[root@localhost AutoVendor]# cd ..
[root@localhost jerry]# cd ..
[root@localhost root]# cd workspace/
[root@localhost workspace]# ls
asd  AutoSeller  AutoVender  HelloCppUnit  HelloCPPUnitTest  Makefile
[root@localhost workspace]# cd AutoVender/
[root@localhost AutoVender]# ls
AutoVendor.cpp  AutoVendor.h  Debug  main.cpp  Makefile  TestWithCppunit.cpp  TestWithCppunit.h
[root@localhost AutoVender]# make
g++ -DHAVE_CONFIG_H -I /root/jerry/cppunit-1.10.2/config -g -O2 -MD -MP -c \
TestWithCppunit.cpp main.cpp AutoVendor.cpp
bash /root/jerry/cppunit-1.10.2/libtool --mode=link g++ -g -O2 \
-o main main.o TestWithCppunit.o AutoVendor.o /root/jerry/cppunit-1.10.2/src/cppunit/libcppunit.la
mkdir .libs
g++ -g -O2 -o main main.o TestWithCppunit.o AutoVendor.o  /root/jerry/cppunit-1.10.2/src/cppunit/.libs/libcppunit.a
[root@localhost AutoVender]# ./main
.


OK (1 tests)


[root@localhost AutoVender]# 
```

图 5-36　执行测试及测试结果

5.3.3　CppUnit 单元测试应用举例

如果要测试下面这个类

```
class Number
{
    public:
        Number(int);
        ~Number(int);
        int Add(int, int);
        int Sub(int, int);
        int Mul(int, int);
 };
```

的正确性，可以通过编写三个测试用例——testAdd()、testSub()和 testMul()来分别验证其中三个成员函数的正确性。三个测试函数组成一组 TestSuite，而且需要依顺序进行验证。那么，可以定义如下的测试类：

```
class NumberTest : public CppUnit::TestFixture
{
```

```
        CPPUNIT_TEST_SUITE( CSvcObjTest );
        CPPUNIT_TEST( testAdd );
        CPPUNIT_TEST( testSub );
        CPPUNIT_TEST( testMul );
        CPPUNIT_TEST_SUITE_END();
    public:
        // setup & teardown
        void setUp ();
        void tearDown();
        //real tests functions
        void testAdd();
        void testSub();
        void testMul();
    };
```

其中，testAdd()、testSub()和 testMul()分别为测试 Number 类中三个成员函数的测试用例，并在私有部分规定了这一组 TestSuite 的测试顺序。下面以测试用例 testAdd()的实现来说明测试过程。

```
void NumberTest::testAdd()
{
    Number my_num;
    int a, b;
    CPPUNIT_ASSERT((a+b) ==    my_num.Add(a, b));
    cout<<"Add OK!"<<endl;
}
```

类 Number 的成员函数 Number::Add(int，int)的功能是对两个整型参数进行相加。在测试函数 NumberTest::testAdd()时，当 Number 的成员函数 Add(a，b)的返回值与实际预期值(a+b)不等时，CPPUNIT_ASSERT()就会停止该测试函数的执行，并在屏幕上输出 Failure 信息，定位出出错代码的行号和相关错误内容。

这只是一个简单的 CppUnit 测试方案。在实际运行过程中，可根据功能分配或代码结构分别进行相应的单元测试，并进行多种输入值、边界值的测试，以保证最小单位代码的准确性和健壮性。

5.4　基于标注的单元测试框架 TestNG

TestNG 是一个开源的单元测试框架（它源自于 JUnit 和 NUnit 的启发），其功能可覆盖单元测试（隔离测试一个类）、功能测试、端到端的测试、集成测试（测试由多个类多个包甚至多个外部框架组成的整个系统，例如服务器）。

相比较，JUnit 简单容易理解，但是它的缺陷也是比较明显的，例如，无法向 JUnit 的测试方法传递参数，也无法向 setUp()和 tearDown()方法传递参数；每次执行一个测试方法的时候，都要重新实例化测试类；测试数据只能写死在代码里面，维护和扩充比较困难。

使用 TestNG 框架不需要继承任何类和实现特定的接口，标注(annotations)更加丰富、完善，可使用 annotations 标注 TestNG 指定的测试方法。并且 TestNG 是基于数据驱动的测试框架。可应用到集成测试，可通过 Ant 调用。并且可用于单元级的性能测试。

TestNG 是一个设计用来简化广泛的测试需求的测试框架，从单元测试到集成测试（这个是 TestNG 设计的出发点，不仅是单元测试，而且可以用于集成测试）。设计目标的不同，与 JUnit 只适用于单元测试相比，TestNG 无疑走的更远，可以用于集成测试。

测试过程的三个典型步骤，注意和 JUnit(4.0)相比，多了一个将测试信息添加到 testng.xml 文件或者 build.xml 文件中。这样，测试信息尤其是测试数据不再写死在测试代码中，好处就是修改测试数据时不需要修改代码/编译了，从而有助于将测试人员引入单元测试/集成测试。

基本概念，相比 JUnit 的 TestCase/TestSuite(JUnit 中的 TestCase 将 test/test method 混合，比较容易让人概念不清晰，尤其是新手)，TestNG 有 suite/test/test method 三个级别，即将 test/test method 明确区分开了。另外，TestNG 中用到 annotation 的快速预览，还有它们的属性。

5.4.1　TestNG 功能介绍

由于 TestNG 是基于 J2SE 5.0 的标注特性所构建的，因此必须了解标注的一些基本概念。

1. 标注概念

标注是 J2SE 5.0 所新提供的对于元数据的支持。程序开发人员可以在不改变原有逻辑的情况下，在源文件嵌入一些补充的信息。标注都是由@Interface annotationName 来声明的。标注可以用来修饰类定义、方法、域变量等。使用的时候是在修饰的对象的定义前@annotationName。标注可以包含多个属性，使用的时候为属性赋值，例如@annotationName(prop1=value1,prop2=value2)。程序的开发人员还可以通过 Java 的反射特性，在运行时获得这些标注的信息。在后面会看到 TestNG 是如何使用它所定义的标注类型来实现测试框架的。

2. 测试步骤

在 TestNG 中编写一个测试通常分为三个步骤。

(1) 编写测试业务逻辑，并且在代码中插入 TestNG annotations。此时，在 TestNG 中要用到 annotation 的快速预览，还有它们的属性。

(2) 在 testng.xml 或者 build.xml 添加测试信息。例如，类名、希望运行的测试集等。

(3) 运行 TestNG。

5.4.2　TestNG 环境建立

TestNG 需要运行在 JDK 1.4 以上，因为 TestNG 使用 Java 中的 annotations。

1. 安装 JDK

下载安装 JDK。配置相应的环境变量，检查 JDK 环境。

在 DOS 窗口中，输入命令“java –version”查看安装的 JDK 版本。输入“javac”命令查看结果，判断环境变量是否配置正确，如图 5-37 所示。

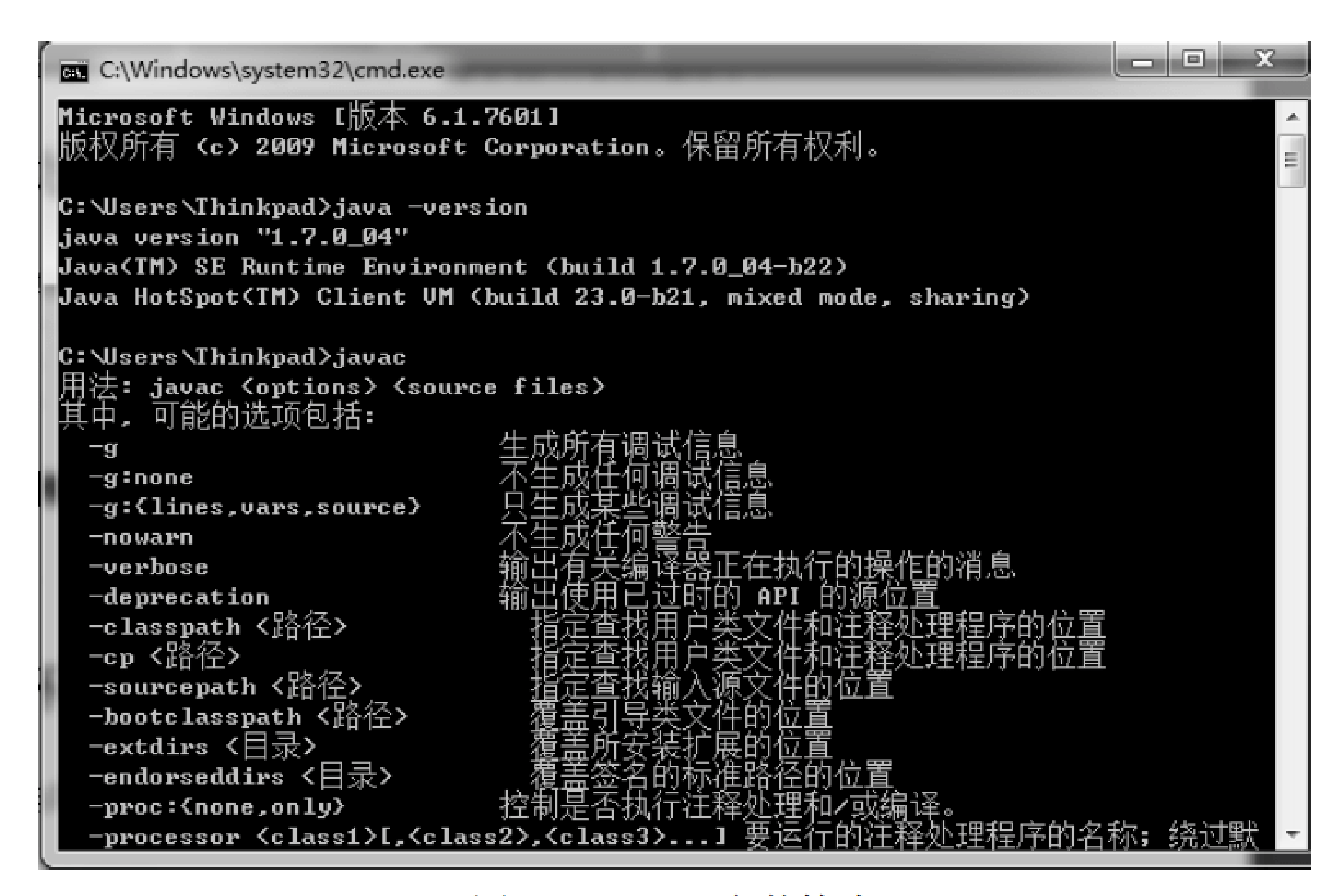


图 5-37　JDK 安装检查

2. IDE 中配置 TestNG

1) 下载安装 Eclipse

安装完毕之后，打开 Eclipse。

在 Eclipse 中配置安装 TestNG 的步骤如下。

(1) 单击 Eclipse 里菜单栏的 Help|Install New Software 命令，如图 5-38 所示。

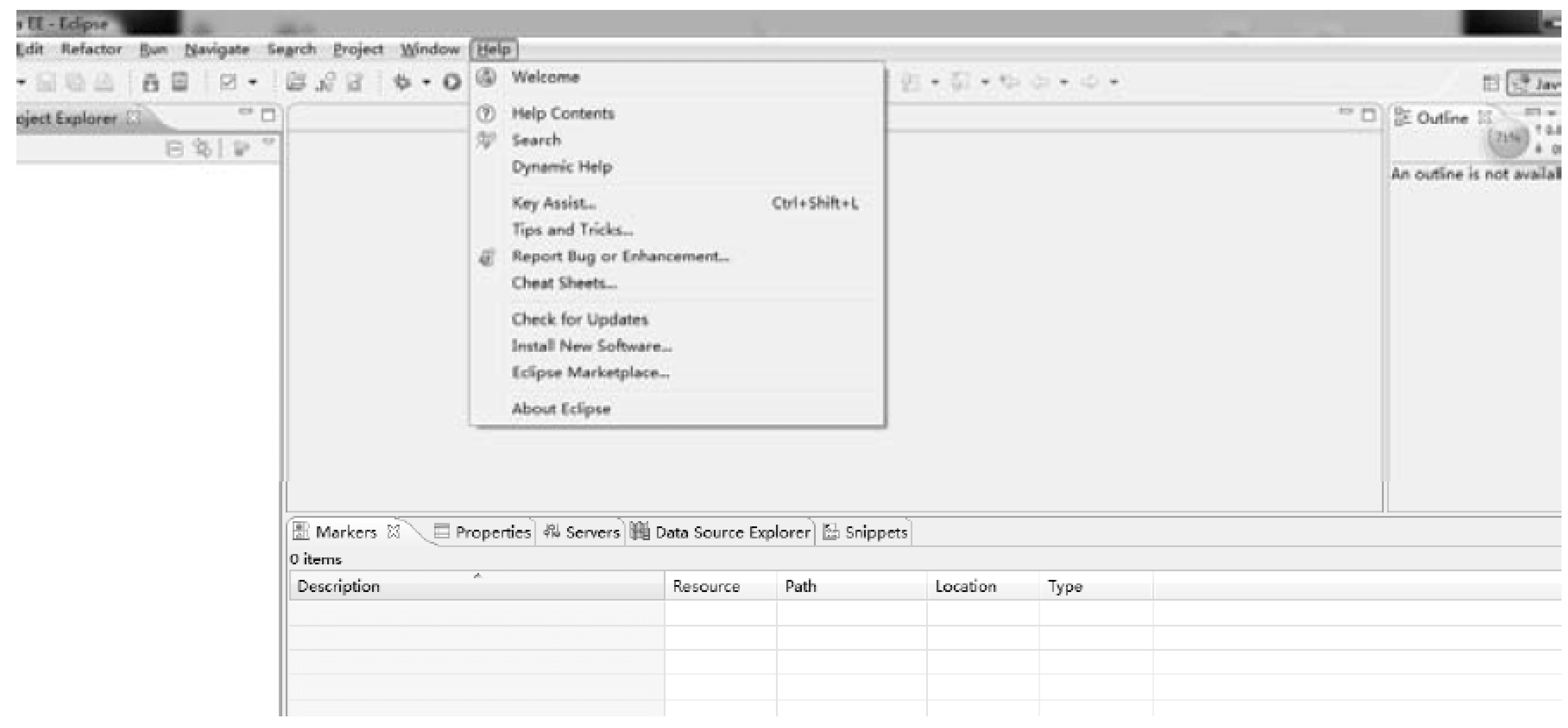


图 5-38　新软件安装

(2) 单击 Add 按钮，如图 5-39 所示。

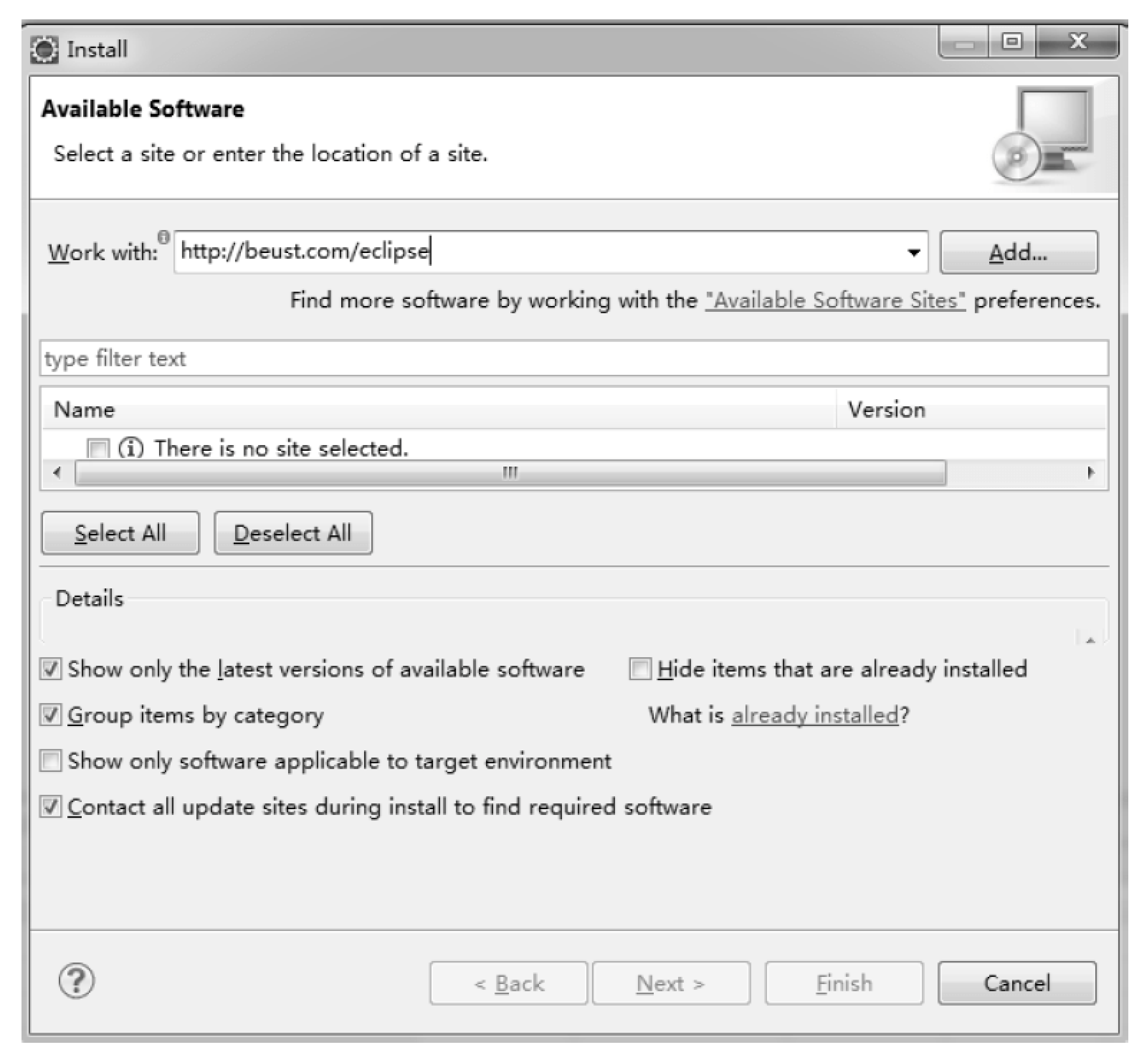


图 5-39　单击 Add 按钮

(3) 在弹出的窗口中输入“http://beust.com”，结果如图 5-40 所示。

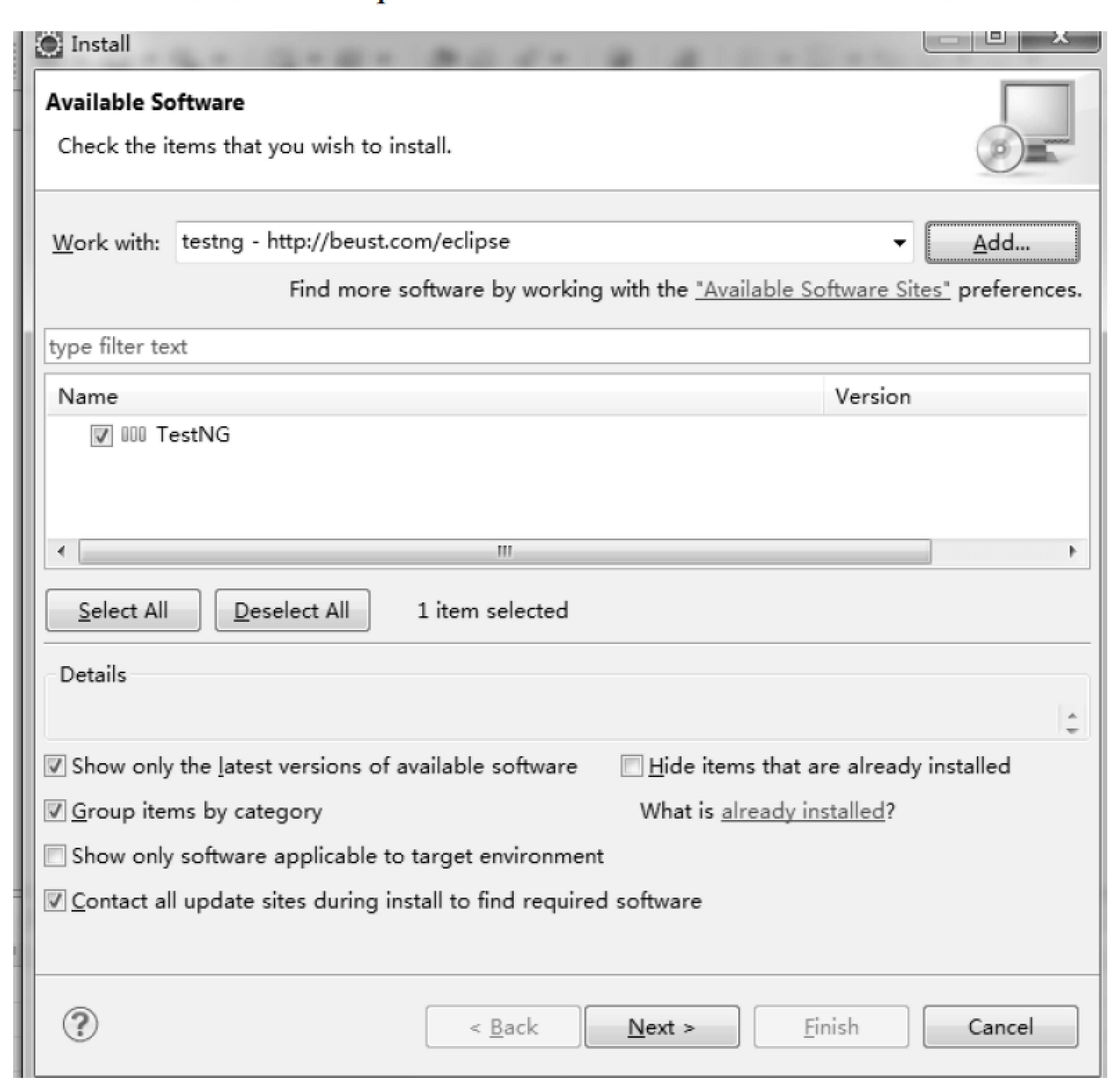


图 5-40　输入“http://beust.com”后的结果

(4) 选择 TestNG 复选框，单击 Next 按钮，按照提示安装，安装完毕之后重启 Eclipse。

2) 在要测试的 Java 项目中加入 TestNG 的 jar 包

在要进行测试的项目上右击，选择 Build Path|Add Library 命令，如图 5-41 所示。此时，项目结构如图 5-42 所示。

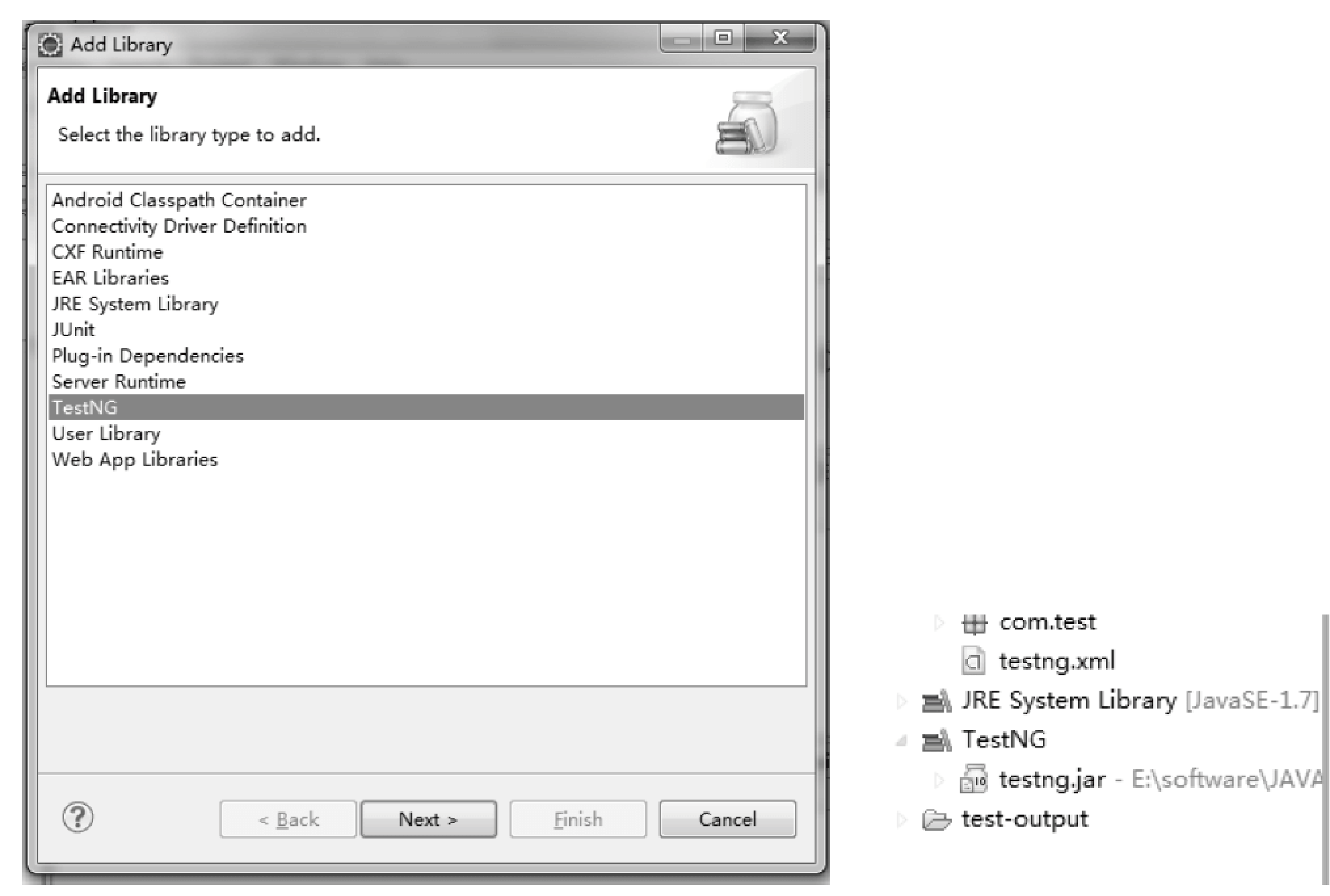


图 5-41　加入 TestNG 的 jar 包　　　　图 5-42　项目结构图

5.4.3　TestNG 应用流程

TestNG 和 JUnit 不同，它使用标注、正则表达式和基于 XML 的配置文件对测试方法进行配置。

1. 工作流程

可以根据一个小例子来具体解释 TestNG 的工作流程，具体如下。

(1) 在 Eclipse 中创建一个 Java 的项目 com.catherine.lab.testng.demo。

(2) 在 Packet Explorer 中，右键单击刚生成的项目，选择 Properties 命令。

(3) 在 Properties 属性框中，选择 Java Build Path，单击 Add External JARs…。

(4) 在文件浏览的对话框中，选择

{eclipse 3.X home directory}/plugins/com.beust.testng.eclipse_XXX/eclipse_testng.jar

和

{eclipse 3.X home directory}/plugins/com.beust.testng.eclipse_XXX/lib/testng-jdk14.jar

以及

testng-jdk15.jar

单击 OK 按钮。

(5) 在 Project 中创建一个 package: com.catherine.lab.testng.firstTest。在 package 里边创建一个类：FristTestSample。

下面是 TestNG 的一个例子（代码 1）。

```
package com.catherine.lab.testng.firstTest;
import com.beust.testng.annotations.*;
public class FirstTestSample {
        public FirstTestSample() {
            super();
        }
    @Test
    public void testPass() {
            assert true :  "This test should pass.";
    }

    @Test
    public void testFail() {
            assert false : "This test will fail";
    }

    @Configuration(beforeTestClass = true)
        public void doBeforeTests() {
            System.out.println("invoke before test class!");
        }
    @Configuration(afterTestClass = true)
        public void doAfterTests() {
            System.out.println("invoke after test class!");
        }
}
```

(6) 在 Eclipse 中打开 Run|Run...，如图 5-43 所示。首先选择使用 TestNG 的 Project，而后选择编写了测试逻辑的 Class，单击 Run 按钮。测试结果就显示在 TestNG 的视图中了，如图 5-44 所示。

这是一个完整的测试用例。和 JUnit 不同，TestNG 中实现测试逻辑的类不需要继承任何父类。测试方法也无须遵循 test×××的命名规则。

2. 配置过程

TestNG 的类是读者所非常熟悉的普通的 Java 类，而在这个类中，所有的被@Test 这个标注所修饰的方法都会被当作测试方法来运行。除了测试类之外，TestNG 还需要一个配

置文件，用来配置测试过程。以下是一个简单的配置文件：testng.xml。

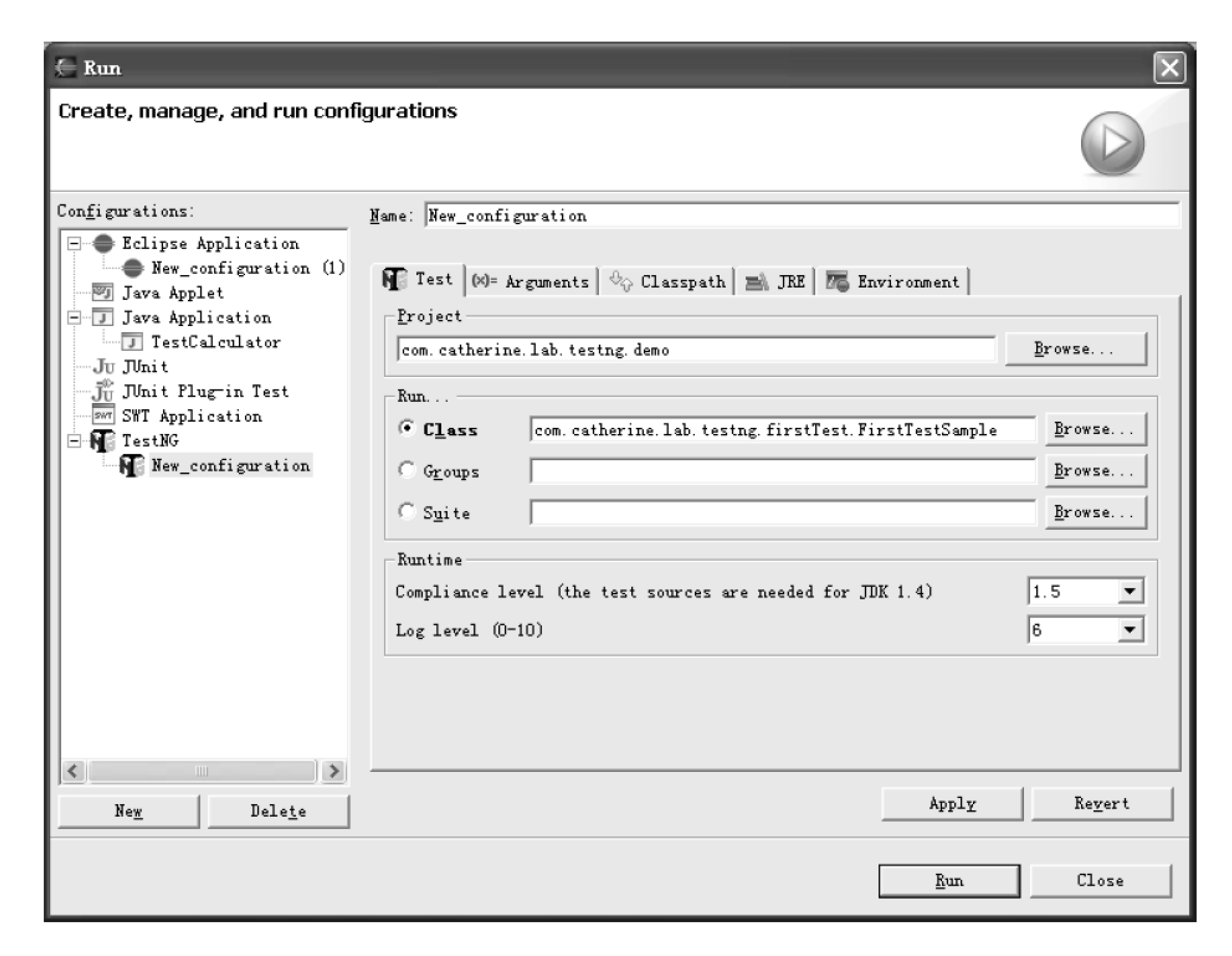


图 5-43　配置运行 TestNG 的程序

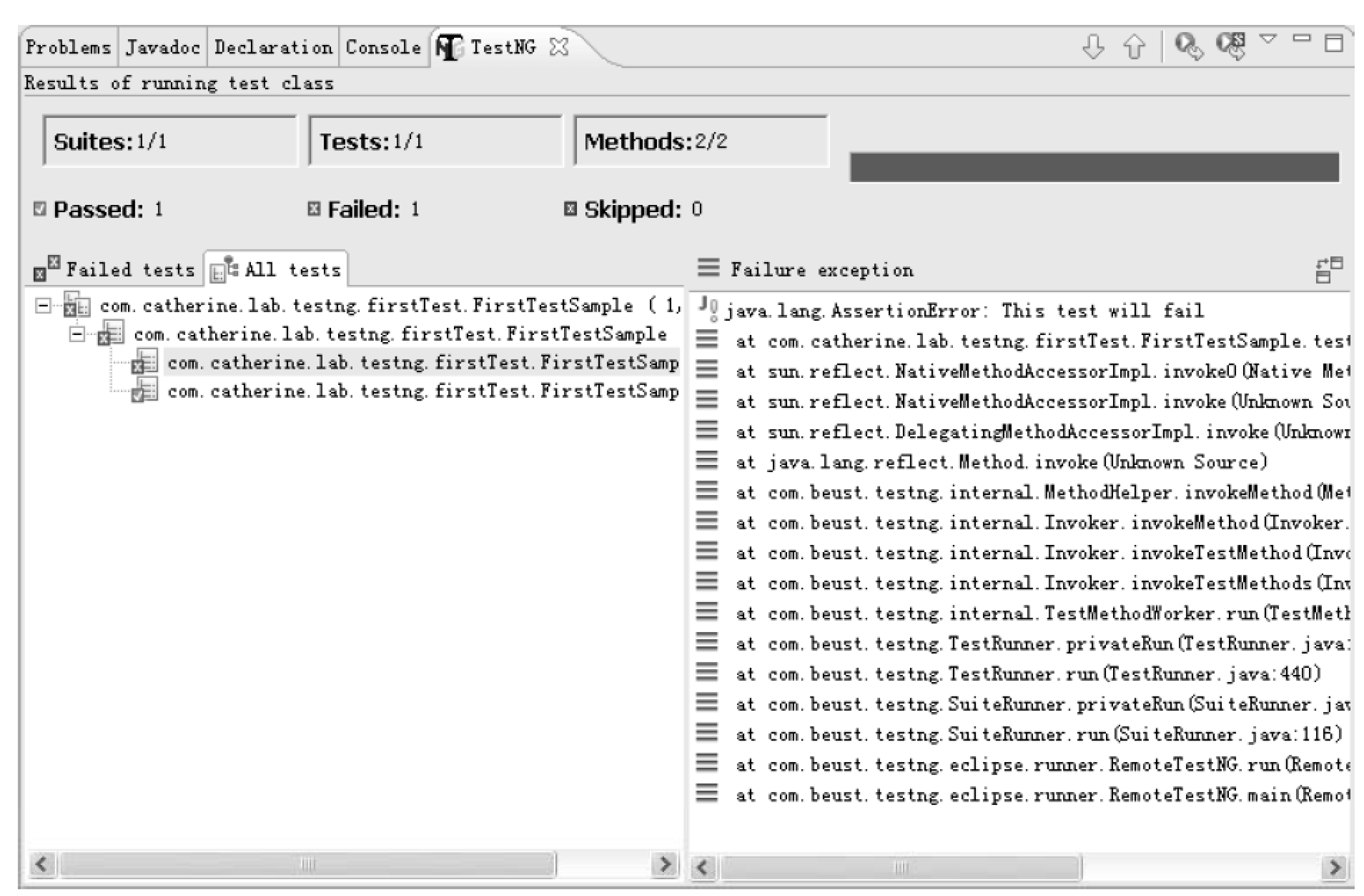


图 5-44　TestNG 的运行结果

下面是 TestNG 的配置文件（代码 2）。

```
<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd" >
  <suite name="My First TestNG test">
```

```
    <test name="Hello Test!">
      <classes>
        <class name=" com.catherine.lab.testng.firstTest.FirstTestSample " />
      </classes>
    </test>
</suite>
```

testng.xml 可以配置测试集<suite>，类似于 JUnit 的 TestSuite。而<test>类似于 JUnit 中的 TestCase。所不同的是，TestNG 中的测试集可以包括多个测试用例，一个测试用例可以包括多个测试类，而一个测试类中可以定义多个测试方法。在下面的例子中，将看到比这个配置文件更复杂的应用。

在图 5-43 的运行配置中，也可以设置一个 XML 文件作为配置文件，而不是直接使用测试类。其实在使用测试类的时候，TestNG 也生成了一个默认的 XML 文件（可以切换到 Resource Perspective，然后刷新 Workspace，就会发现这个 project 里边生成了一个 XML 文件，而这个文件就是 TestNG 的默认的配置文件）。

再回到代码 1 查看，在上面的程序清单中会发现，除了使用@Test 这个标注以外，还使用了@Configuration 这个标注。下面就来介绍@Configuration 这个标注的用途。

在标注 Configuration 中，定义了以下的属性。

下面是 Configuration 中的属性（代码 3）。

```
public boolean beforeSuite() default false;
public boolean afterSuite() default false;
public boolean beforeTest() default false;
public boolean afterTest() default false;
public boolean beforeTestClass() default false;
public boolean afterTestClass() default false;
public boolean beforeTestMethod() default false;
public boolean afterTestMethod() default false;
```

说明如下：

(1) beforeSuite=true，所修饰的方法将在测试套件(也就是配置文件中的 Suite Tag)中任何一个方法调用之前调用一次。

(2) afterSuite=true，所修饰的方法将在测试套件中所有方法都调用过后调用一次。

(3) beforeTest=true，在测试用例（配置文件中 Test Tag）中任何一个测试方法调用之前调用一次。

(4) afterTest=true，在测试用例中任何所有方法都调用之后调用一次。

(5) beforeTestClass=true，在测试类中任何测试方法调用之前调用一次。

(6) afterTestClass=true，在这个测试类中所有方法都调用过后调用一次。

(7) beforeTestMethod=true，在每个测试方法调用之前调用一次。

(8) afterTestMethod=true，在每个测试方法调用之后调用一次。

代码 1 中 doBeforeTests()方法，在任何一个 test 方法调用之前被调用一次。doAfterTests，就是所有的 test 方法运行过了以后再调用一次。从 Console 输出的信息中，可以验证这一点，如图 5-45 所示。

```
======
TESTCLASS: com.catherine.lab.testng.firstTest.FirstTestSample
[TestClass] BeforeClass : com.catherine.lab.testng.firstTest.FirstTestSample.doBeforeTests()
[TestClass] Test        :               com.catherine.lab.testng.firstTest.FirstTestSample.testPass()
[TestClass] Test        :               com.catherine.lab.testng.firstTest.FirstTestSample.testFail()
[TestClass] AfterClass  : com.catherine.lab.testng.firstTest.FirstTestSample.doAfterTests()
[TestClass]
======
```

图 5-45　Console 输出的运行信息

3. 异常检测

使用 TestNG，可以非常简单、非常容易地检测异常的发生。很明显，用 JUnit 也可以做这件事，但是正如下面示例中所看到的，使用 TestNG 的@ExpectedExceptions 标注可以使代码编写惊人地容易和简单。@ExpectedExceptions 标注指明框架能够容忍引发的 NumberFormatException 异常，所以不应当被当作是故障。要查看在某行代码中是否引发异常，可以直接在这行代码之后加入 assert false 语句。这意味着只有在指定行中引发特定类型的异常的时候，才会通过测试。

```
public class  NumberUtilsTest
 {
      @Test(groups = {"tests.math"})
      @ExpectedExceptions(NumberFormatException.class)
      public void test()
      {
           NumberUtils.createDouble("12.23.45");
           assert false;   //shouldn't be invoked
      }
 }
```

5.4.4　TestNG 应用举例

5.4.3 节中介绍了使用 TestNG 的一个最简单的例子，这一节中将介绍一些关于 TestNG 的高级应用。标注 Test 除了标志其修饰的方法为测试方法，还提供了 groups 的属性。

比如上面例子的两个方法 testPass()和 testFail()，可以给这两个方法加上 group 的属性。

1. 测试组定义

测试@Test 的 groups 属性（代码 4）。

```
@Test(groups={"functional_test"})
    public void testPass() {
```

```
        assert true :   "This test should pass.";
    }
        @Test(groups={"checkin_test"})
    public void testFail() {
        assert false : "This test will fail";
    }
```

而后打开 Run|Run…，在配置文件的 Runtime 配置中选择 Groups，然后选择要运行的 group 的名字，如图 5-46 所示。

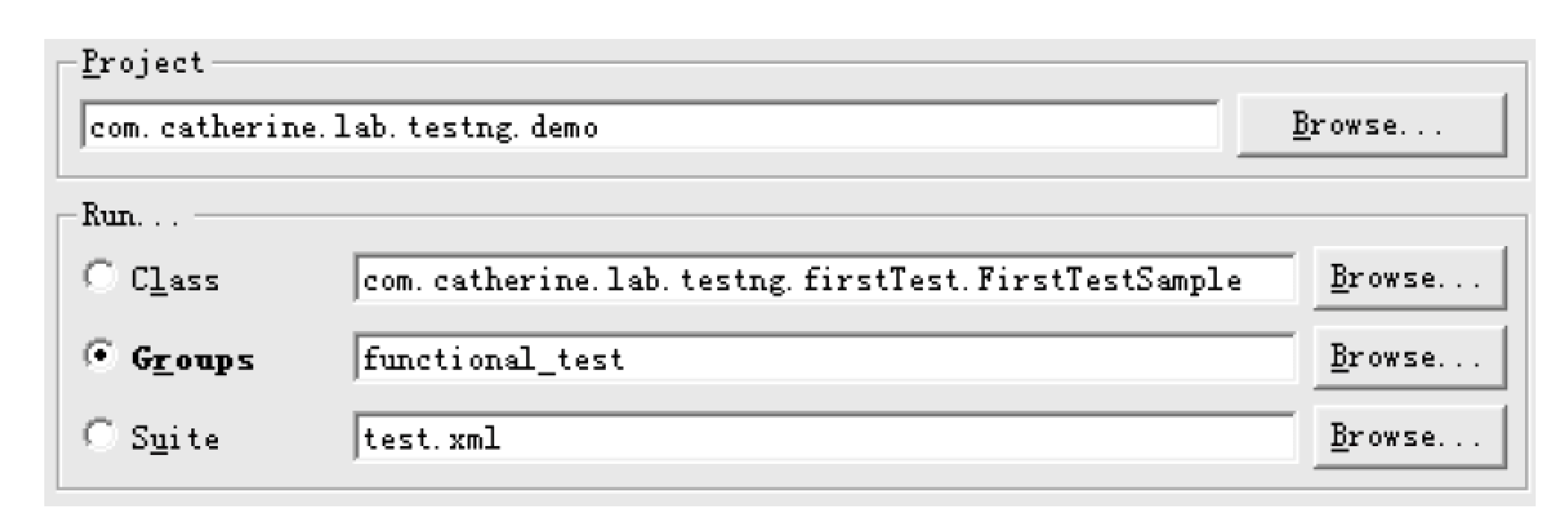


图 5-46　运行选定的测试组

此时，从 TestNG 中看到测试结果，只有 testPass 运行了，而 testFail 因为不属于 funcational_test 这个组，因此并没有运行，如图 5-47 所示。



图 5-47　运行的结果

和第一个例子类似，虽然在这里并没有显式地定义配置文件，testNG 已经生成了相应的配置文件了。在 Resource Perspective 底下可以看到这个文件：Custom_Suite××××.xml。

2. 测试@Test 的属性

自动生成的配置文件（代码 5）。

```
<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd" >
<suite name="My First TestNG test">
    <test name="Hello Test!">
     <groups>
        <run>
         <include name="functional_test"/>
        </run>
     </groups>
        <classes>
         <class name=" com.catherine.lab.testng.firstTest.FirstTestSample " />
        </classes>
    </test>
</suite>
```

除了 groups 属性以外，标注 Test 还支持属性 dependsOnMethods 和属性 dependsOnGroups，这两个属性主要用于规定测试方法的执行顺序。

TestNG 并不保证按照定义的顺序执行测试方法。如果这些测试方法之间有依赖关系，那么就可以使用 dependsOn××××的属性。还是看第一个例子，现在在这个例子里边增加了一个方法：

```
setupEnvforPass()
```

我们希望 setupEnvforPass（）方法在 testPass 方法前执行，修改了 testPass 的 test 标注，如代码 6 所示。

测试@Test 的属性 dependsOnMethods（代码 6）。

```
@Test(groups={"functional_test"},dependsOnMethods = { "setEnvForPass" })
     public void testPass() {
            assert true :   "This test should pass.";
     }
     @Test(groups={"checkin_test"})
     public void testFail() {
            assert false : "This test will fail";
}
@Test(groups = {"init"})
public void setEnvForPass(){
          assert true: "This is dependent method"
}
```

运行配置和配置文件都不需要改动，现在来运行这个例子，测试结果如图 5-48 所示。可以看到，虽然在 testPass 方法之后定义了 setEnvForPass 方法，但是由于将 setEnvForPass 定义为 testPass 的依赖方法，setEnvForPass 在 testPass 前执行了。

同样，可以定义 dependsOnGroups 的属性，这样只有 Groups 中所有的方法都被执行完，

这个方法才会被执行。注意：如果 depensOnGroups 中制定的 group 在配置文件中被 excluded 了，那么这个方法会依然被执行。但是如果指定的 group 在配置文件中被 include 了，而 group 中的方法有错误，那么这个方法会被 skip，不会被执行。

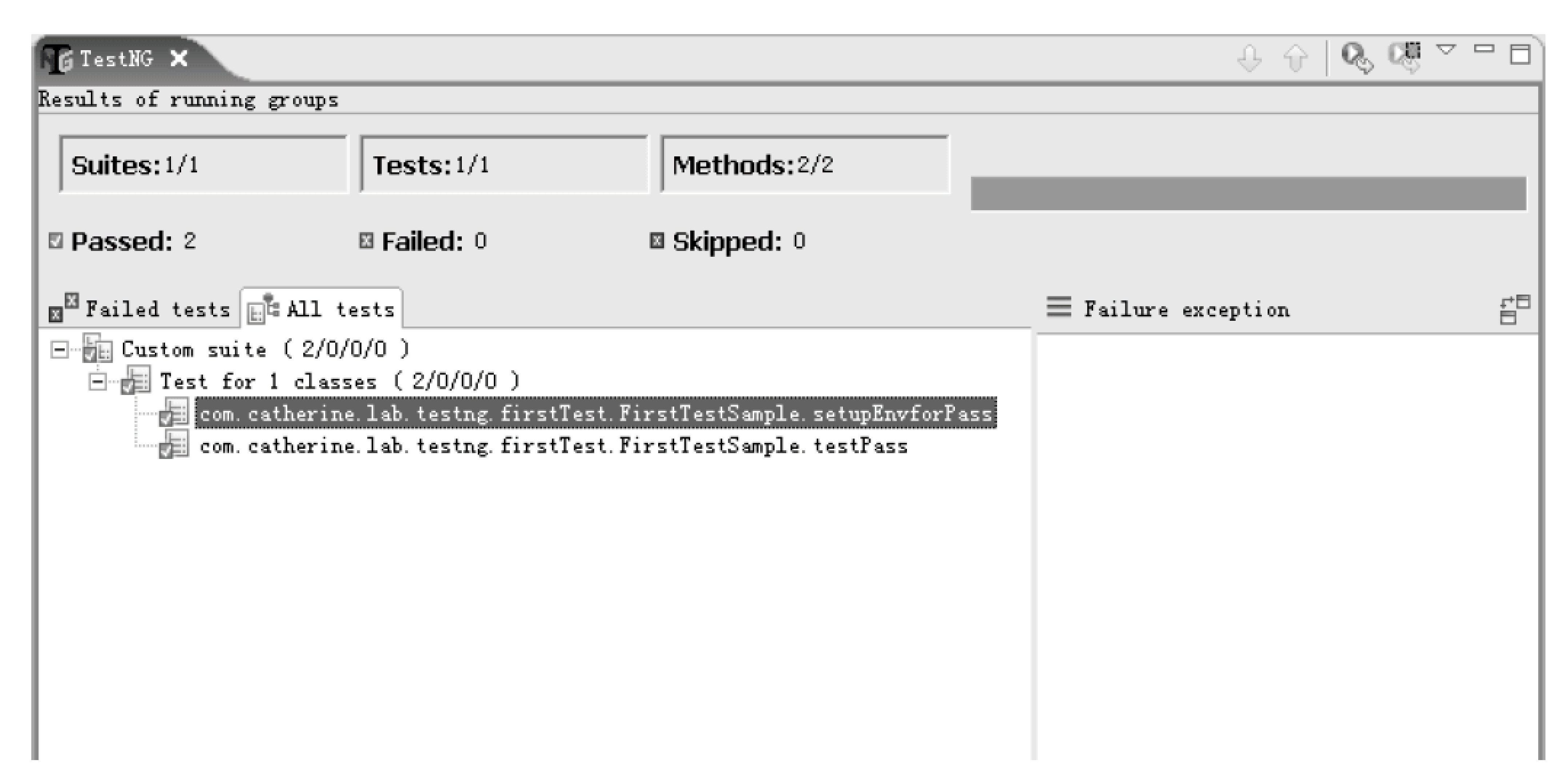


图 5-48　运行结果

3. 测试标注@Parameter

下面介绍一个新的标注类型：@Parameter。

TestNG 的测试方法可以带有参数，参数可以通过@Parameter 来声明，具体的参数值在 testng.xml 中定义。这是 TestNG 的一个很优越的特性。还是在以前的例子上的基础上来验证这个特性。为 setEnvForPass 这个方法定义一个参数 target_server，并且在测试方法中打印这个参数。

测试标注 Parameter（代码 7）。

```
@Parameters({ "target_server" })
@Test(groups = {"init"})
public void setEnvForPass(String targetServer){
    assert true: "This is dependent method";
System.out.println(targetServer);
}
```

Target_server 的值在 testng.xml 中定义。在 TestNG 的运行时配置中选择 Suite,然后选择清单 8 中定义好的 testng.xml。运行 TestNG，从 Console 的运行结果中看到，target_server 的值被打印出来了。

自定义的配置文件（代码 8）。

```
<!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd">
<suite name="Custom suite">
  <parameter name="target_server"   value="127.0.0.1"/>
```

```
    <test verbose="6" name="Test for 1 classes" annotations="1.5">
      <groups>
        <run>
          <include name="functional_test"/>
        </run>
      </groups>
      <classes>
        <class name="com.catherine.lab.testng.firstTest.FirstTestSample">
        </class>
      </classes>
    </test>
</suite>
```

测试方法的参数可以是任意多个，只要通过配置文件传入了正确的参数，那么测试方法中就可以使用这些参数了。不过需要注意的是，参数是有作用域的，比如参数可以在配置文件的 suite 和 test 之后定义，而如果两个参数的名称一样，test 中定义的参数值有较高的优先级。

TestNG 可以从多个线程中运行测试方法，只需要将配置文件中 suite 的 parallel 属性设为 true。线程的数目在 thread-count 中设置。如果两个方法有依赖关系，那么它们将在一个线程中运行，除此之外，都可以在多个线程中并发地运行。

```
<suite name="My suite" parallel="true" thread-count="10">
```

4. 其他特性

(1) TestNG 提供了标注 Factory，用来动态生成测试方法的参数。

(2) TestNG 还提供了 AntTask <testng>，可以从 Ant 脚本中调用 testNG。

(3) TestNG 可以调用 JUnit 的 test cases。只需要在配置文件中声明：

```
<test name="Test1" junit="true">
<classes ..>
```

(4) TestNG 还可以从程序中调用：

```
TestNG testng = new TestNG();
testng.setTestClasses(Class[] {});
testng.run();
```

5.4.5　TestNG 与 JUnit4 对比

JUnit 和 TestNG 在外表上是相似的,但总体来说 TestNG 比 JUnit4 功能要丰富一些。JUnit 主要针对的是单元代码的测试，TestNG 适合高水平的测试；另外，TestNG 能够很好地支持 Selenium 页面自动化测试工具的使用；最后，基于 JUnit 存在着大量的插件，以及

其他业界产品如 Ant、Maven 以及 Eclipse 等广泛地支持着 JUnit，使得 JUnit 仍然是首选。表 5-14 给出了 TestNG 和 JUnit4 在各种特性和指标方面的比较结果。

表 5-14　TestNG 与 JUnit4 在各种特性和指标方面的比较情况

特性或指标	TESTNG	JUint4
1. 测试是否支持(Annotations)	YES	YES
2. 框架依赖程度	不需要扩展特定的基类和实现特定的方法	不需要扩展特定的基类和实现特定的方法
3. 灵活性	同样支持 Before、After 方法，TestNG 更为灵活，支持各种签名方式，如 private、protected。同样也支持 BeforeClass 和 AfterClass，只执行一次的方法，但是可以不需要使用 static 签名	支持 Before、After 方法 支持 BeforeClass 和 AfterClass
4. 依赖性测试	利用 Test 注释的 dependsOnMethods 属性来应对测试的依赖性问题。有了这个便利的特性，就可以轻松指定依赖方法。	难以确定测试用例执行的顺序
5. 失败和重运行	一旦 TestNG 中出现失败，它就会创建一个 XML 配置文件，对失败的测试加以说明。如果利用这个文件执行 TestNG 运行程序，TestNG 就只运行失败的测试。这样可以快速定位出错方法，并且节约大量的时间。 失败文件，一般命名为 testng-failed.xml，以后只需要运行此文件就可以了	如果测试套件包括 N 项测试，其中 $M(M \leqslant N)$ 项失败，很可能就会迫使用户重新运行整个测试套件（修改错误以后）。这样的工作会耗费掉大量的时间
6. 参数化测试	TestNG 提供了开箱即用的类似特性。通过在 TestNG 的 XML 配置文件中放入参数化数据，就可以对不同的数据集重用同一个测试用例，甚至有可能会得到不同的结果。 支持@DataProvider，注释可以方便地把复杂参数类型映射到某个测试方法中	如果想改变某个受测方法的参数组，就只能给每个不同的参数组编写一个测试用例。多数情况下，这不会带来太多麻烦。出现大量的重复测试代码
7. 测试分组	支持	不支持
8. 多线程测试	TestNG 对多线程测试的支持良好，只需要配置即可	JUnit 中要想进行多线程测试比较麻烦，需要其他模块

实 验 习 题

1. 应用 xUnit 中的其他单元测试工具(如 nUnit)，并给出它们详细的应用流程。
2. 搜索并下载其他类型的开源单元测试工具，与 xUnit 进行全面比较。

第6章　单元覆盖测试

覆盖测试是衡量软件质量的一个重要指标，它是一种“白盒”测试方法，或者说是“白盒”测试的主要内容。覆盖的标准有逻辑覆盖、循环覆盖和基本路径覆盖。其中，逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。这些覆盖发现错误的能力呈由弱至强变化：语句覆盖要求每条语句至少执行一次；判定覆盖要求每个判定的每个分支至少执行一次；条件覆盖要求每个判定的每个条件应取到各种可能的值；判定/条件覆盖要求同时满足判定覆盖和条件覆盖；条件组合覆盖要求每个判定中各条件的每一种组合至少出现一次；路径覆盖要求程序中每一条可能的路径至少执行一次。

覆盖测试要求测试人员必须拥有程序的规格说明和程序清单，以程序的内部结构为基础来设计测试用例。其基本准则是以测试用例来尽可能多地覆盖程序的内部逻辑结构，发现其中的错误和问题。所以，覆盖测试一般应用在软件测试的早期，即单元测试阶段。

在单元测试中，开发人员或测试人员很清楚程序的内部工作过程，能全面了解程序内部的逻辑结构，因此可通过测试来检测程序内部动作和逻辑结构是否按照规格说明书的规定正常进行。按照程序内部的逻辑结构设计测试数据，按照覆盖要求测试程序，检验程序中的每条通路是否都能按预定要求正确工作。此时功能的正确性被放到了次要地位。

覆盖测试目前主要用在具有高可靠性要求的软件领域，例如军工软件、航天航空软件、工业控制软件等。而这些领域的软件规模都比较大，代码较复杂，逻辑判断较多，因此要取得较好的覆盖测试效果，需要借助一定的工具软件。这些工具软件一般具备如下的功能特点，可弥补人为测试的缺陷。

(1) 分析软件内部结构，帮助制定覆盖策略及设计测试用例。

(2) 与适当的编译器结合，对被测软件实施自动插桩，以便在其运行过程中生成覆盖信息并收集这些信息。

(3) 根据收集的覆盖信息计算覆盖率，帮助测试人员找到未被覆盖的软件部位，以改进测试用例、提高覆盖率。

(4) 对主流开发语言的支持，因为对于不同的开发语言来说，测试工具实现的方式和内容差别是较大的。目前测试工具主要支持的开发语言包括：标准 C、C++、Visual C++、Java、Visual J++等。

在利用工具进行动态覆盖测试时，需要完成三项工作：设计测试用例、对被测代码进行插桩、收集覆盖信息并进行分析。代码插桩由工具自动完成，通过执行测试用例，再由工具收集覆盖信息并进行分析，就可以看到覆盖率指标了。

不同的测试工具对于代码的覆盖能力也是不同的，通常能够支持判定/条件覆盖的测试工具其价格是极其昂贵的。覆盖测试工具在选购时应当注意主要是对开发语言的支持、代码覆盖的深度、覆盖测试结果的可视化等。

6.1　覆盖测试工具介绍

xUnit 尽管号称是以“白盒”测试为主的工具，但实质上从它所提供的单元测试框架来看，它基本上不支持“白盒”测试中最重要的测试内容——覆盖测试。这是因为对于不同的语言来说，覆盖测试工具实现的技术方式和内容差别是很大的，很难有一个统一的实现框架或实现模式。例如，在 JUnit 框架下通过 EclEmma 插件方式实现了单元覆盖测试功能(JCoverage 也能实现覆盖测试功能)，这是因为 Java 语言是解释语言的缘故；在 NUnit 框架下通过提供 NCover 插件方式也能实现单元覆盖测试功能，其原因和 Java 语言类似；而在 CppUnit 单元测试框架中却很难提供覆盖测试功能，这是因为 C/C++的覆盖测试是采取插桩的技术方式来实现的。因此，对于不同的语言，开展覆盖测试的策略是不一样的。

商用的覆盖测试工具有很多，功能也很完整和成熟。广州凯乐软件技术有限公司开发的 Visual Unit 是一款功能实用、物美价廉，而且学习资源非常丰富的国产单元测试工具。而针对 Java 语言的开源覆盖测试工具也有很多，针对 C/C++语言的开源覆盖测试工具中比较著名的是与 GCC 配套的 Gcov 工具。通过学习和使用这些工具，能够很有效地帮助我们学习和掌握覆盖测试的知识和技能。

6.2　JUnit 下的覆盖测试工具 EclEmma

现在 IT 开发人员比以往任何时候都更加关注测试的重要性，没有经过良好测试的代码更容易出问题。在极限编程中，测试驱动开发已经被证明是一种能有效提高软件质量的方法。在测试驱动的开发方式中，软件工程师在编写功能代码之前首先要编写测试代码，这样便能从最开始保证程序代码的正确性，并且能够在程序的每次演进时都进行自动的回归测试。测试对于软件产品的成败起着至关重要的作用，在极限编程领域，甚至有人提议任何未经测试的代码都应该自动从发布的产品中删除。尽管这种说法太过极端，但是测试本身的质量确实是一个需要高度关注的问题。测试的覆盖率是测试质量的一个重要指标，我们需要通过工具来帮助我们对软件测试覆盖结果进行考察。

EclEmma 就是这样一个能帮助开发人员进行覆盖测试的优秀的 Eclipse 开源插件。EclEmma 在覆盖测试领域是非常受欢迎的，它曾在 2006 年入围 Eclipse Community Awards Winners 决赛。虽然最后失败，但从这也可以看出 EclEmma 对开发人员确实能够提供很大的帮助。

6.2.1　EclEmma 介绍

提到 EclEmma，首先就要说到著名的 Java 覆盖测试工具——Emma。Emma 是一个在 SourceForge 上进行的开源项目。从某种程度上说，EclEmma 是 Emma 的一个图形界面。

Emma 的作者在开发 Emma 之初，程序员就已经有了各种各样优秀的开源 Java 开发工具。举例来说，有优秀的集成开发环境 Eclipse，有开源的 JDK，有单元测试工具 JUnit，有 Ant 这样的项目管理工具，还可以用 CVS 或 SubVersion 来进行源代码版本的维护。当时看来，也许唯一缺少的就是一个开源的覆盖测试工具了。Emma 就是为了填补这项空白而生的。现在的情况已经和 Emma 诞生的时候不一样了。时至今日，已经有了不少的覆盖测试工具。例如，Coverlipse 是一个基于 Eclipse 的覆盖测试插件，其他的还有 Cobertura、Quilt 和 JCoverage 等。但是，Emma 因具有一些非常优秀的特性而使得它更适合被广泛地使用。和 Coverlipse 等工具比起来，Emma 是开源的，同时它对应用程序执行速度的影响非常小。

EclEmma 的出现弥补了 Emma 用户一个大的遗憾——缺乏图形界面以及对集成开发环境的支持。将 Eclipse 和 Emma 这两个在各自领域最为优秀的工具结合起来，这就是 EclEmma 所提供的。

总之，EclEmma 是一个基于 Emma 的免费的 Java 代码覆盖工具。它的目的是让用户可以在 Eclipse 工作平台使用强大的 Java 代码覆盖工具 Emma。EclEmma 是非侵入式的，不需要修改项目或执行其他任何安装，它能够在工作平台中启动，并像运行 JUnit 测试一样直接对代码覆盖进行分析。覆盖结果将立即被汇总并在 Java 源代码编辑器中高亮显示。EclEmma 具有如下特点：快速的开发和测试周期，非常丰富的覆盖信息分析以及非入侵的测试方式。

6.2.2　EclEmma 测试环境建立

EclEmma 插件的安装和其他大部分 Eclipse 插件的安装过程相同，既可以通过 Eclipse 标准的 Update 机制来远程安装 EclEmma 插件(如图 6-1 所示)，也可以通过从站点上下载 zip 文件并解压到 Eclipse 所在的目录来安装。

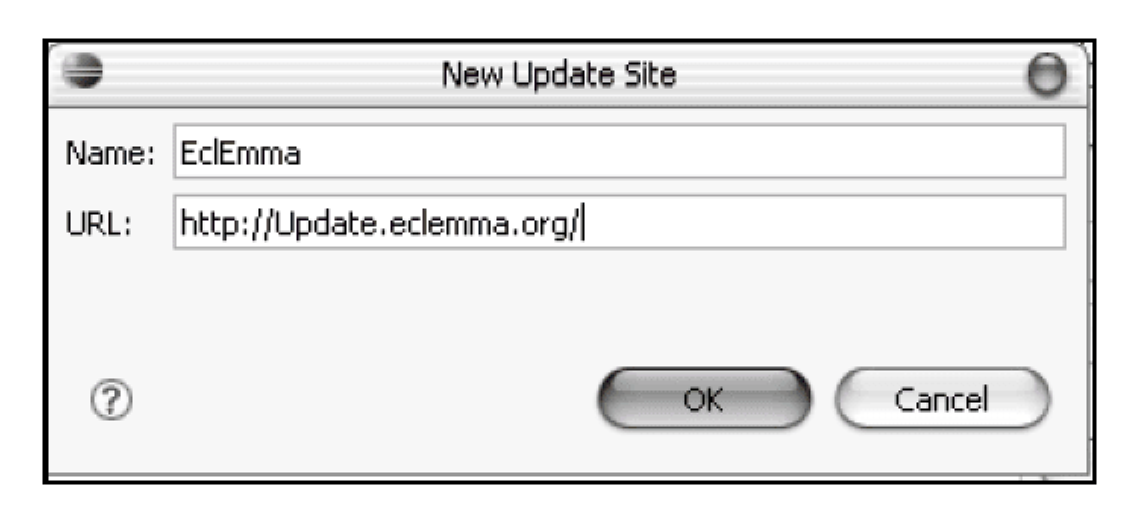


图 6-1　添加 EclEmma 更新站点

不管采用何种方式来安装 EclEmma，安装完成并重新启动 Eclipse 之后，工具栏上都

应该出现一个新的按钮，如图 6-2 所示。

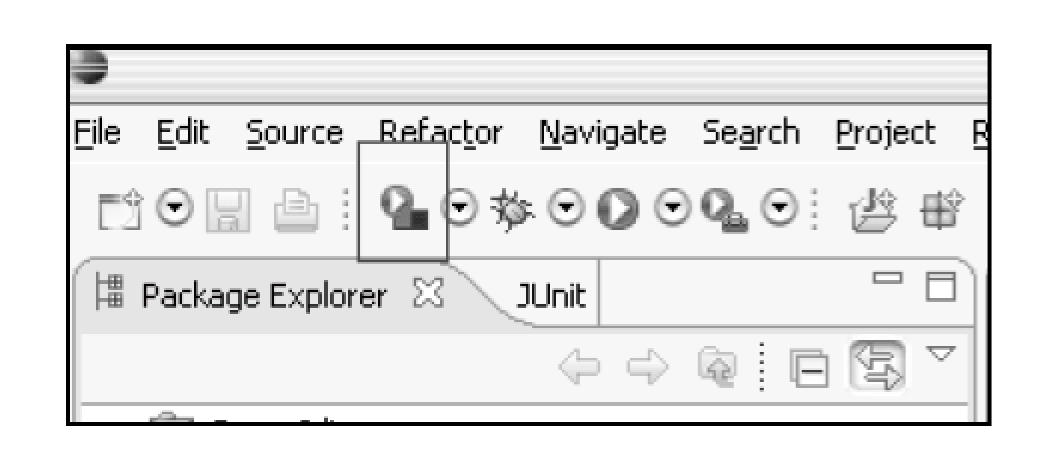


图 6-2　新增的覆盖测试按钮

可按如下步骤来下载并直接安装 EclEmma。

(1) 从 http://sourceforge.net/projects/eclemma 下载 EclEmma 的安装压缩包（当前最新版本为 EclEmma 2.2.10）。

(2) 将压缩包解压缩，可以分别看到 eclemma-2.2.10 这个文件夹中有名为 features 和 plugins 的文件夹。

(3) 找到 Eclipse 在计算机中的安装位置，打开 Eclipse 安装文件，将会看到这个里面也会有相应的 features 和 plugins 这两个文件夹。

(4) 将文件夹 eclemma-2.2.10 中的 features 和 plugins 文件夹中的内容复制到 Eclipse 安装文件中相应的 features 和 plugins 文件夹中。

(5) 关闭 Eclipse 并重新启动。

(6) 在 Eclipse 的界面中将会看到新增的覆盖测试按钮。

6.2.3　EclEmma 测试功能及使用流程

1. EclEmma 的测试功能

1) 不同的颜色表示不同的测试情况

在 Java 编辑器中，EclEmma 用不同的颜色标识源代码的测试情况。其中，绿色的行表示该行代码被完整执行，红色的行表示该行代码根本没有被执行，而黄色的行表明该行代码只有部分被执行。黄色的行通常出现在单行代码包含分支的情况中，如图 6-3 所示(参见彩图)。

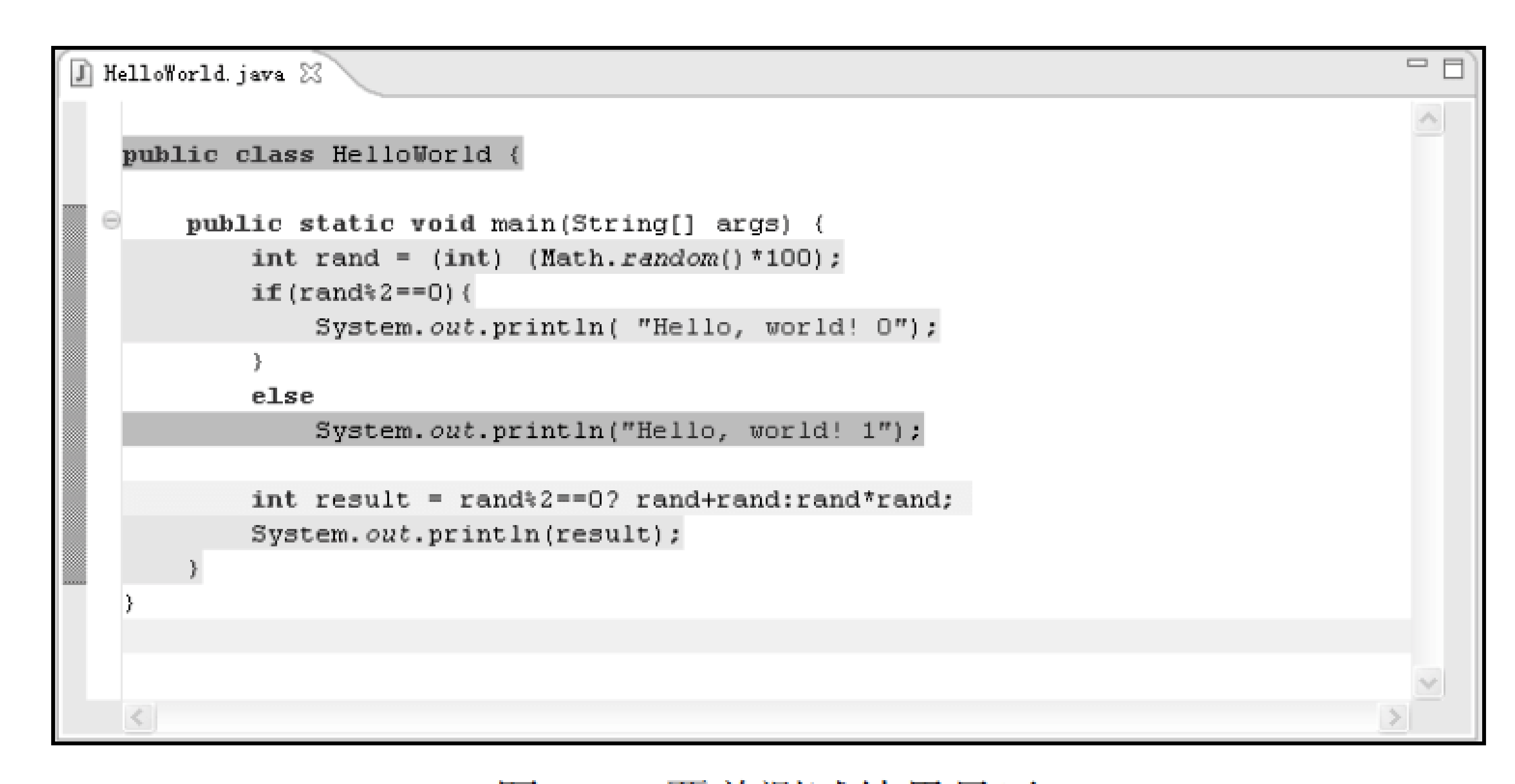


图 6-3　覆盖测试结果显示

2) 分层显示代码的覆盖测试率

EclEmma 提供的 Coverage 视图能够分层显示代码的覆盖测试率。图 6-4 中的信息表明对 HelloWorld 的一次运行覆盖了大约 68.6%的代码。

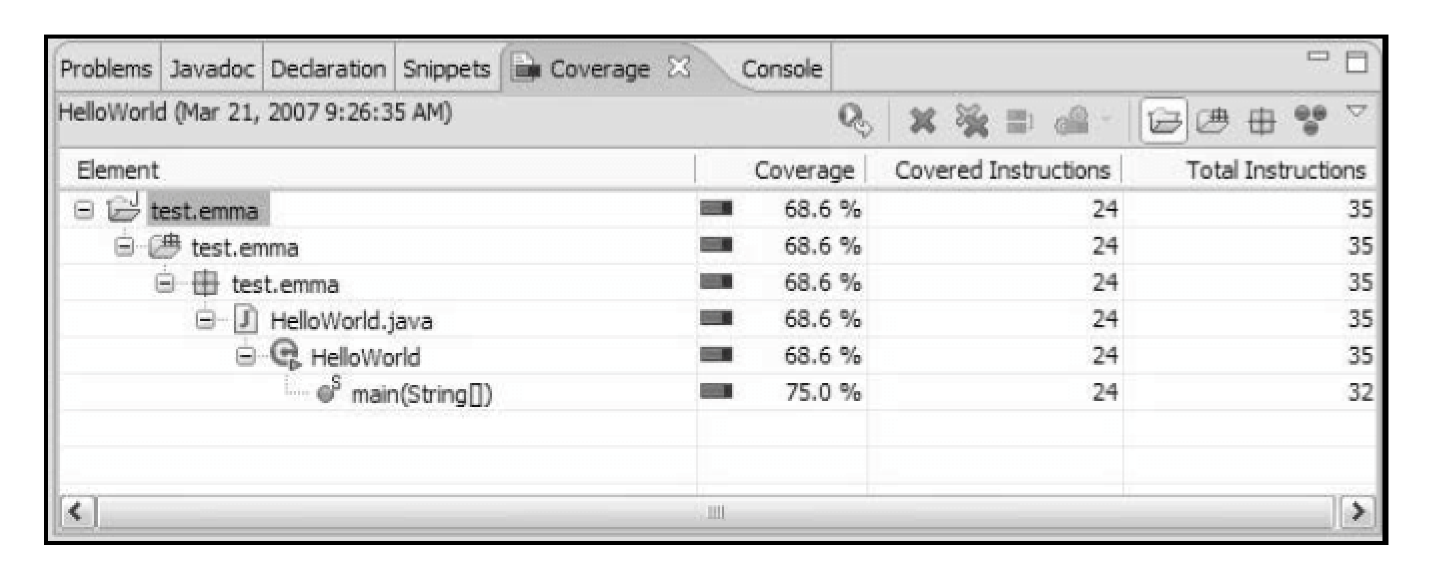


图 6-4 覆盖率统计

3) 可以合并多次覆盖测试的结果

想在一次运行中覆盖所有的代码通常会比较困难，如果能把多次测试的覆盖数据合并起来进行查看，那么就能更方便地掌握多次测试的测试效果。EclEmma 提供了这样的功能。

EclEmma 保存了所有的测试结果。通过 Coverage 视图的工具按钮就可以结合多次覆盖测试的结果，如图 6-5 所示。

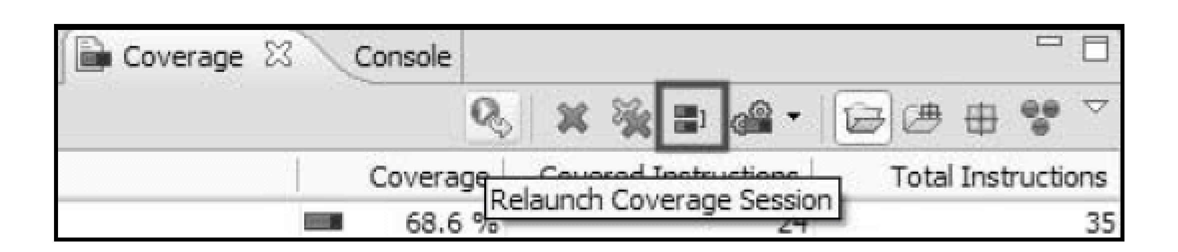


图 6-5 合并多次覆盖测试的结果

2. EclEmma 的使用流程

(1) 建立测试项目。为了实验 EclEmma 的特性，首先在 Eclipse 的 Workspace 中建立一个名为 test.Emma 的新 Java 项目，如图 6-6 所示。

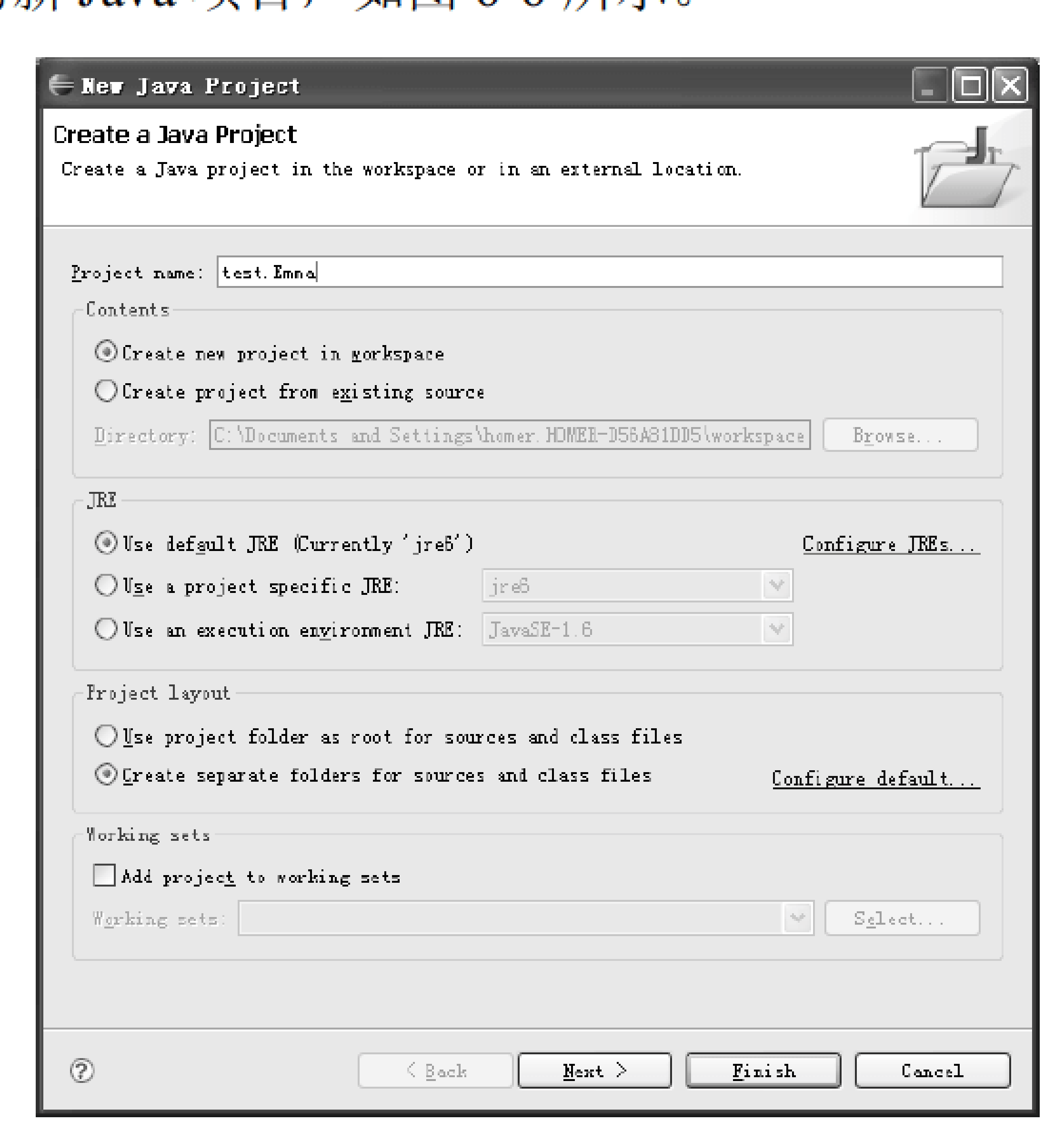
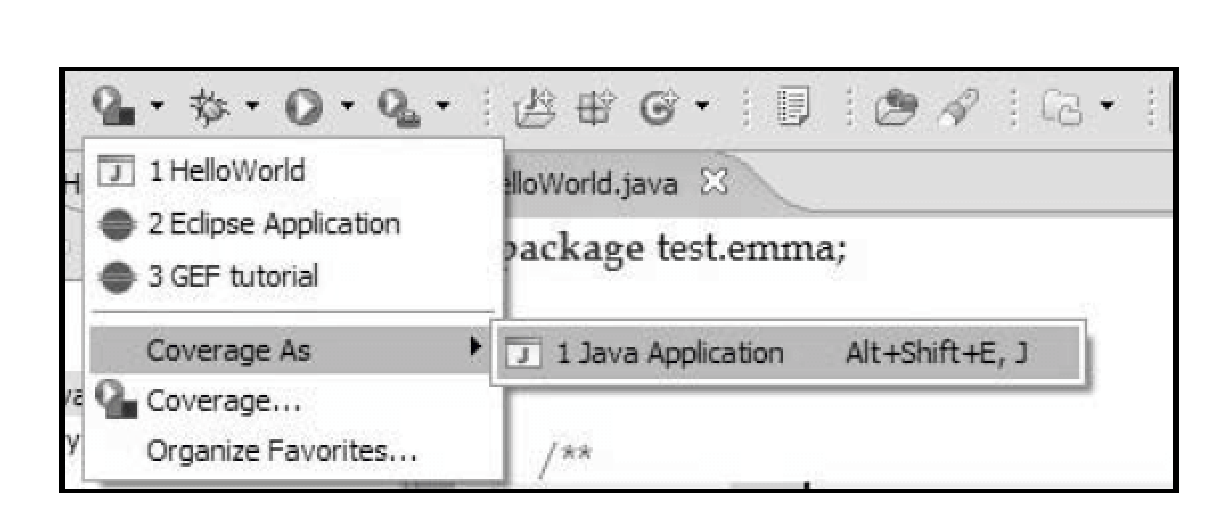


图 6-6 新建测试项目

(2) 编写用于测试 EclEmma 的代码，并在 test.Emma 这个 Java 项目的默认包中建立一个 HelloWorld 类：

```
package test.emma;
public class HelloWorld {
/**
  * @param args
  */
public static void main(String[] args) {
        int rand = (int) (Math.random()*100);
        if(rand%2==0){
                System.out.println( "Hello，  world! 0");
        }
        else
                System.out.println("Hello，  world! 1");
        int result = rand%2==0? rand+rand:rand*rand;
        System.out.println(result);
  }
}
```

(3) 对 Java 应用程序进行覆盖测试，如图 6-7 所示。



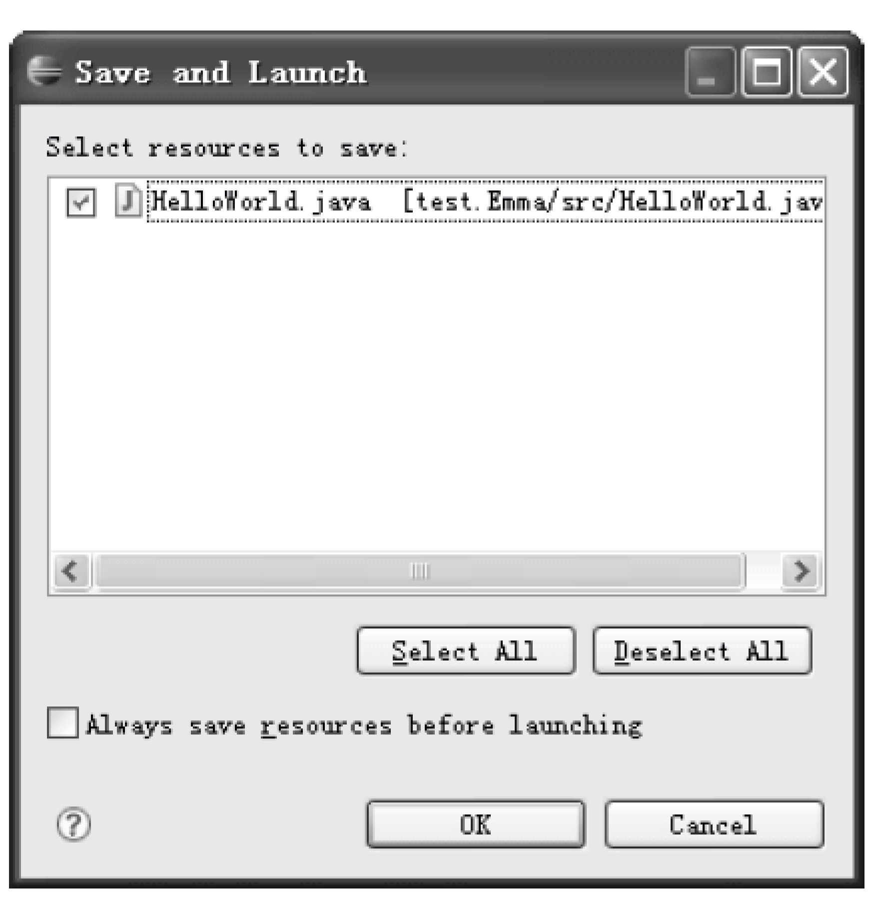


图 6-7　执行覆盖测试

覆盖测试执行完毕之后，正在编辑的 HelloWorld.java 的窗口将会变成如图 6-3 所示效果。

(4) 选择需要合并的覆盖测试结果，如图 6-8 所示。

(5) 查看合并后的覆盖测试结果，如图 6-9 所示(参见彩图)。

通过图 6-9 可以看到，通过多次运行覆盖测试，最终代码达到了 91.4%的测试覆盖率。有趣的是，图中第三行代码被标记为红色，而此行代码实际上是不可执行的。奥妙在于，没有生成任何 HelloWorld 类的实例，因此默认构造函数没有被调用，而 EclEmma 将这个

特殊代码的覆盖状态标记在类声明的第一行。

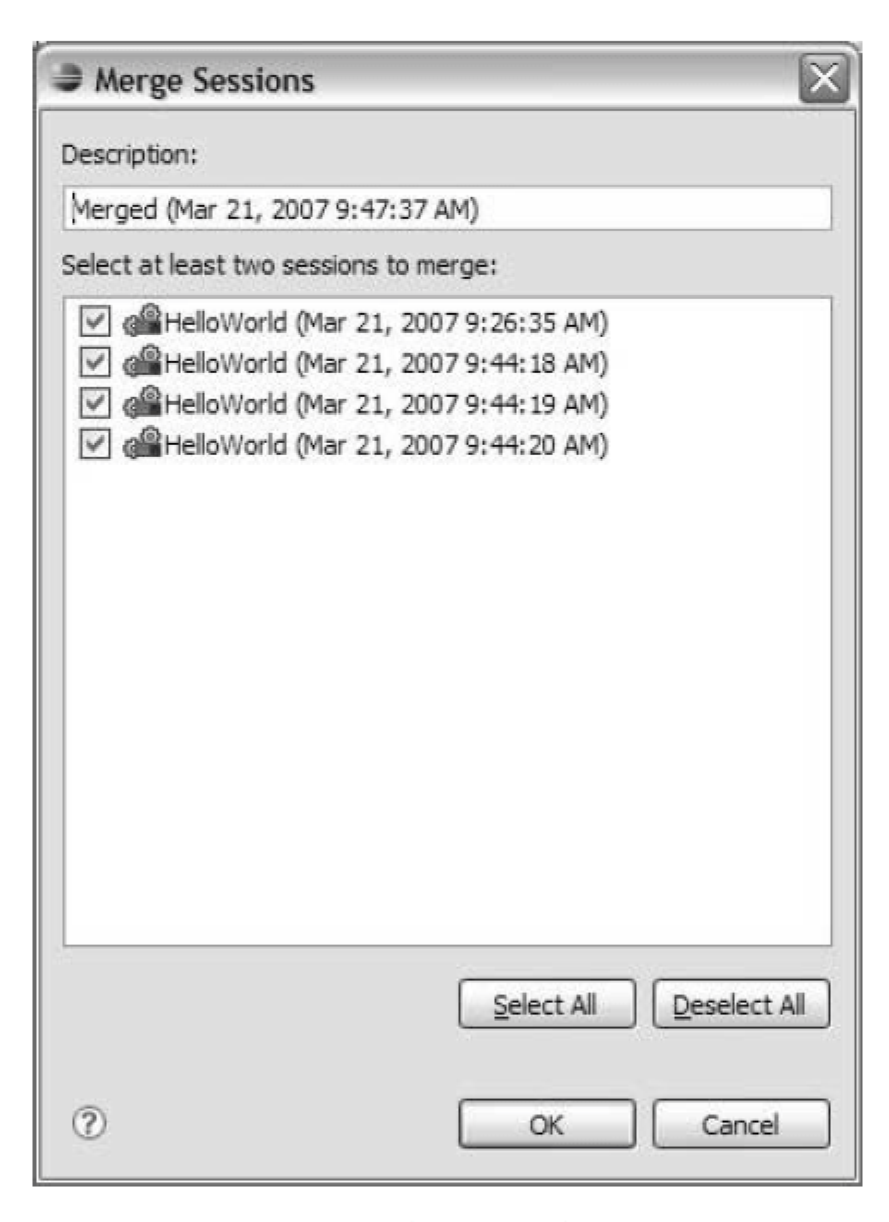


图 6-8　合并覆盖测试

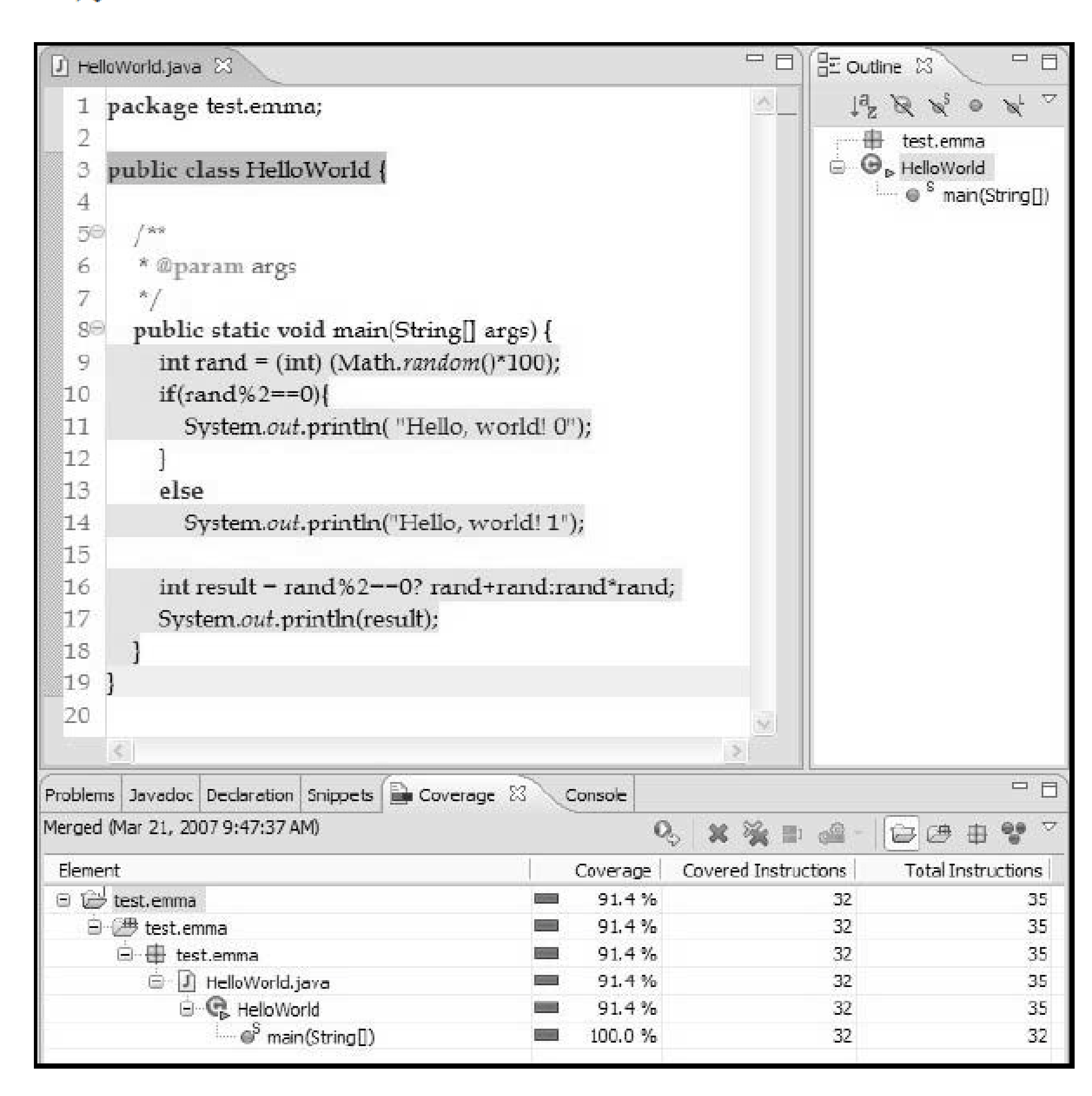


图 6-9　覆盖测试合并结果

3. EclEmma 的高级特性

如果 EclEmma 只能测试 Java 应用程序的测试覆盖率，那么相对命令行版本的 Emma 来说，它提供的增强功能就不多了。相反，EclEmma 提供了很多与 Eclipse 结合紧密的功能。它不仅能测试 Java 应用程序，还能计算 JUnit 单元测试、对 Eclipse 插件测试的覆盖率。从图 6-10 中可以看出 EclEmma 目前支持 4 种类型的程序。

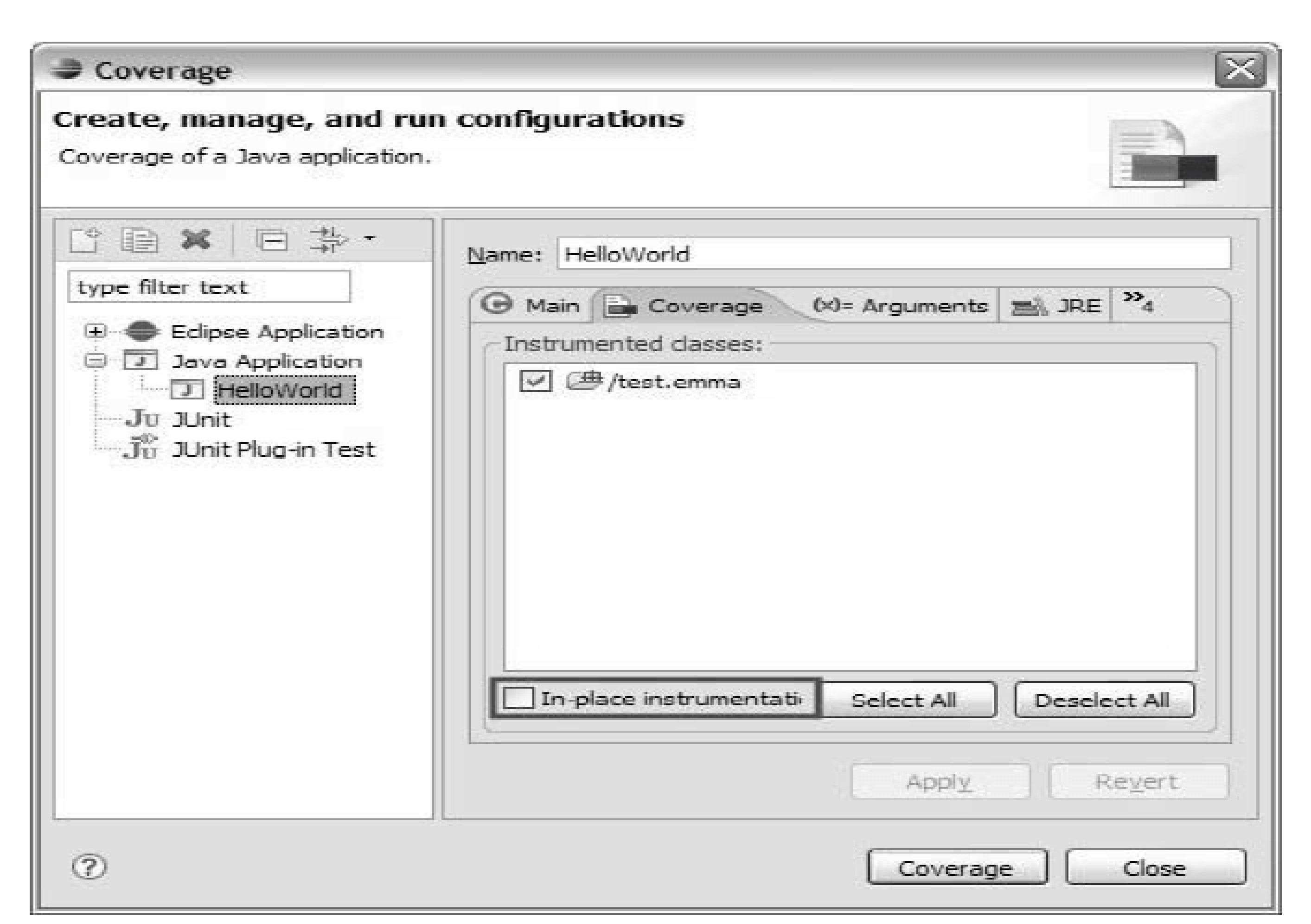


图 6-10　EclEmma 的配置界面

为了进一步了解 EclEmma 是如何获得覆盖测试数据的，需要先对 Emma 有个初步的了解。通常代码覆盖测试工具都需要对被执行的代码进行修改，Emma 提供了以下两种方式来完成这件事。

(1) 预插入模式：对程序进行测试之前，需要用 Emma 中的工具对 class 文件或 jar 文

件进行修改。修改后的代码可以立刻被执行。覆盖测试的结果将会被存放到指定的文件中。

(2) 即时插入模式：即时插入模式不需要事先对代码进行修改。相反，对代码的修改是通过一个 Emma 定制的 Class loader(类载入器)进行的。这种方式的优点很明显，不需要对 class 文件或 jar 文件进行任何修改。缺点是为了获得测试的结果，需要用 Emma 提供的命令 emmarun 来执行 Java 应用程序。

使用即时插入模式的优点很明显：class 文件和 jar 文件不会被修改。而预插入模式的应用范围更为广泛，对于某些需要嵌入到框架中运行的代码来说(例如 EJB)，只能使用预插入模式。EclEmma 仅使用了 Emma 的预插入模式来工作，不过 EclEmma 默认会在临时目录中创建 class 文件和 jar 文件的副本来进行修改，因此在 workspace 中 class 和 jar 文件仍然保持原样。虽然听上去很好，但是由于需要修改 classpath 来使用修改过的 class 文件和 jar 文件，对于不能修改 classpath 的应用(例如 Eclipse RCP 和 JUnit Plugin Test)来说，还是只能选择修改 workspace 中的 class 文件和 jar 文件。对于 Java Application 和 JUnit 类型的覆盖测试来说，可以在配置对话框中选中 In-place instrumentation 来指定直接修改 workspace 中的.class 文件和.jar 文件。

6.2.4　EclEmma 测试应用举例

本节仍以售货机程序为例，来进行覆盖测试。

这个程序的设计原理是这样的：顾客投入硬币来购买自己想要的饮料；已设定好了两种饮料——beer 和 orange；每种饮料的价格是 5 角钱；将两种饮料的数量设定为 3 个，5 角钱和 1 元钱的数量分别是 3 个。下面来看看售货机在不同状态下的行为。

(1) 如果顾客投入的是正好的 5 角钱，那么可以选择自己所要的饮料，beer 或是 orange。分别显示如下两条信息："You have pay for the beer. Please pick it up."和"You have pay for the orange. Please pick it up."。

(2) 如果选择的 beer 或是 orange 的数量超过了设定的 3 个，那么售货机会报错，提示顾客饮料已售光，将显示信息："There has no beer，please pick up your money，Sorry!"。顾客将拿回自己的钱，得不到饮料。

(3) 如果顾客选择的是这两种饮料以外的其他饮料，那么系统将提示错误，因为只存在这两种饮料。

(4) 如果顾客投入的不是 5 角钱而是 1 元钱，那么顾客可以在饮料有剩余且有零钱找的情况下得到饮料及找回的零钱，显示如下信息："You have pay for the beer. Please pick it up and the loose change."或是"You have pay for the orange. Please pick it up and the loose change."。

(5) 如果顾客投入 1 元钱，但是饮料已经售完了，那么顾客将拿回自己的钱，得不到饮料。

(6) 如果顾客投入 1 元钱，饮料也有，但是没有零钱找给顾客，那么同样顾客将拿回自己的钱，得不到饮料。这时会显示："There has no loose change，Please pick up your money，Sorry!!!!!"。

(7) 如果顾客投入的既不是 5 角钱也不是 1 元钱，那么售货机肯定无法找零钱，于是报错，显示信息："There has some input error!!!!!"。

1. 详细代码

```
public class SaleMachine {
        private int _beerNum，_orangeNum，_count_of_five，_count_of_one;
        private float _total;
        private String[] _type={"beer"，"orange"};
        private String _result;
        public SaleMachine()
        {
                init();
        }
        /*public SaleMachine(int five，int beer，int orange)//便于测试的初始化函数
        {
                 _count_of_one=one;
                _count_of_five=five;
                _beerNum=beer;
                _orangeNum=orange;
        }*/
        private void init()
        {
        _beerNum=3;
        _orangeNum=3;
        _count_of_five=3;
        _count_of_one=5;
}
public String Operate(String type，int money)
//type 是用户选择的产品，money 是用户投币的种类
{
        float loose_change=0;
        if(money= =5)   //如果用户投入 5 角钱
        {
                if(type.equals(_type[0]))      //如果用户选择啤酒
                {
                        if(_beerNum>=1)//如果还有啤酒
                        {
                                _beerNum– –;
                                _count_of_five++;
                                _result="You have pay for the beer. Please pick it up.";
```

```
                System.out.println(_beerNum);
                return _result;
            }
            else
                return "There has no beer，please pick up your money， Sorry!";
        }
        else if(type.equals(_type[1])) //如果用户选择橙汁
        {
            if(_orangeNum>=1)
            {
                _orangeNum– –;
                _count_of_five++;
                _result="You have pay for the orange. Please pick it up.";
                System.out.println(_orangeNum);
                return _result;
            }
            else
                return "There has no orange，please pick up your money，Sorry!!";
        }
        else
            return "The type message is errno!!!";
    }
    else if(money= =1)      //如果用户投入一元钱
    {
        if(type.equals(_type[0])&&_beerNum>=1)      //如果用户选择啤酒且还有啤酒
        {
            if(_count_of_five>=1)  //如果有零钱找
            {
                _beerNum– –;
                _count_of_five– –;
                _count_of_one++;
                _result="You have pay for the beer. Please pick it up and the loose change.";
                return _result;
            }
            else
                return "There has no loose change，\
                        Please pick up your money，Sorry!!!!";
        }
        else if(type.equals(_type[1])&&_orangeNum>=1)
        //如果用户选择橙汁且还有橙汁
        {
```

```
                if(_count_of_five>=1)  //如果有零钱找
                {
                    _orangeNum--;
                    _count_of_five--;
                    _count_of_one++;
                    _result="You have pay for the orange.\
                            Please pick it up and the loose change.";
                    return _result;
                }
                else
                    return "There has no loose change，\
    Please pick up your money，Sorry!!!!!";
                }
            }
            return "There has some input error!!!!!";
        }
    }
```

2. 用 Eclemma 进行覆盖测试

(1) 建立一个测试类，类名是 salemachineTest。在这个建好的工作界面上，可以看到如图 6-11 所示的几行代码，这是测试类在建立的过程中由软件自动生成的。

```
package SaleMachineTest;

import junit.framework.TestCase;

public class SaleMachineTest extends TestCase {

    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

}
```

图 6-11　生成测试类

在最后的两个大括号之间，就可以编写自己的测试程序了。

(2) 编写测试类的基本步骤。

① 扩展 TestCase 类。

② 覆盖 runTest()方法(可选)。

③ 编写一些 test×××××()方法。

根据上面的基本步骤来编写每一个测试类，就这次测试的 Java 程序来看，大体的测试思路如下。

首先，要保证顾客能够买到自己想要的饮料，也就是说，要把无论顾客投入 5 角钱还

是 1 元钱都能成功买到饮料的程序覆盖测试一遍。要保证这部分的覆盖测试，可以设计如下 4 个测试类：testSaleMachineA(顾客投入 5 角钱能够买到 beer)，testSaleMachineB(顾客投入 5 角钱能够买到 orange)，testSaleMachineC(顾客投入 1 元钱能够买到 beer)，testSaleMachineD(顾客投入 1 元钱能够买到 orange)。可以看到，此时的源程序界面视图变色，被测试覆盖的程序代码变成绿色，没有被覆盖的代码变成红色，如果有黄色的则说明这条语句中没有完全被测试覆盖，如图 6-12 所示是本次测试源程序变色的情况(参见彩图)。

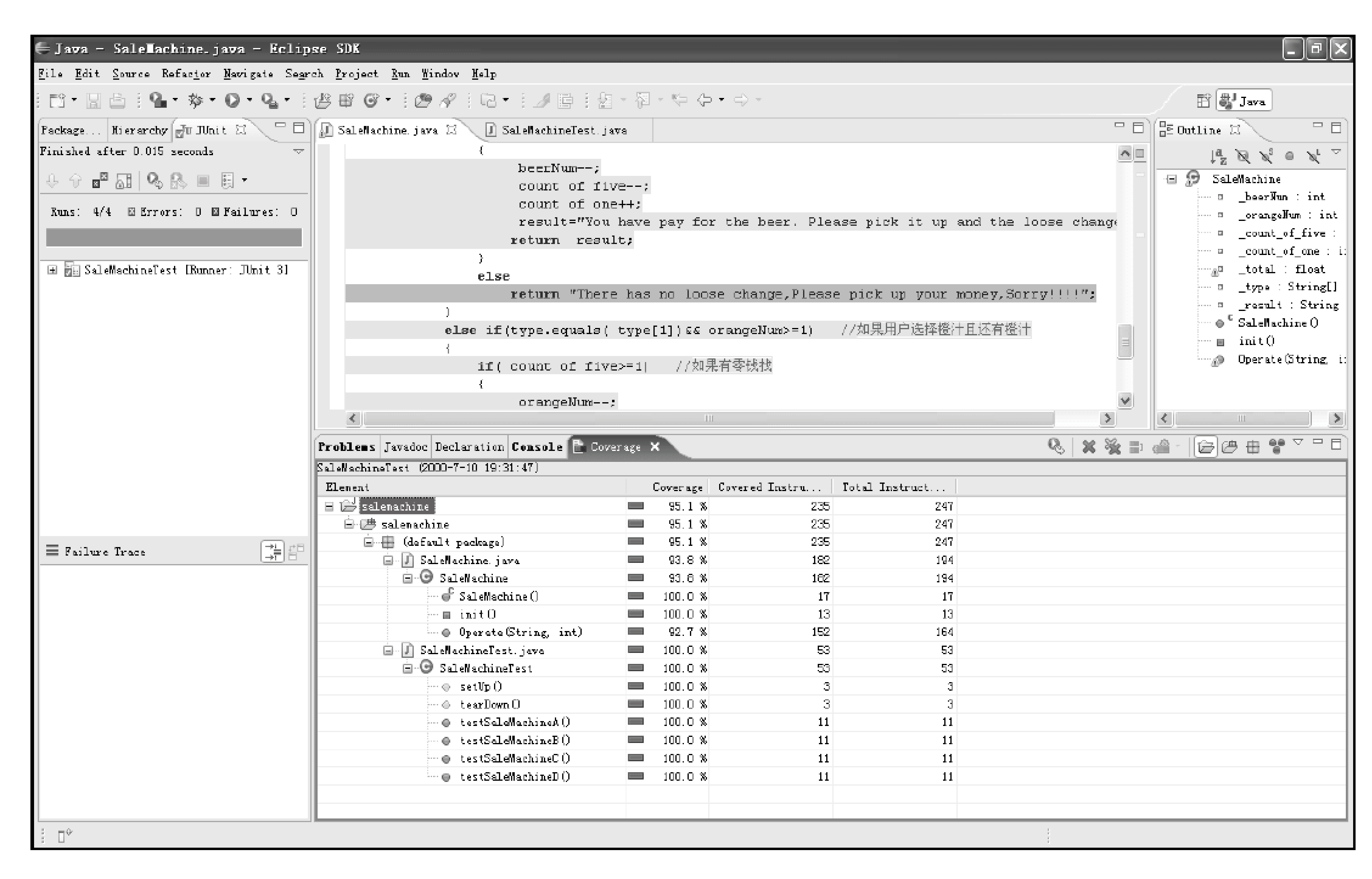


图 6-12　顾客能够买到饮料的覆盖测试结果

右下视图说明在这次覆盖测试中，测试覆盖率是 95.1%。

其次，如果成功执行了顾客可以用 5 角钱或 1 元钱买到 beer 或 orange 的情况，那么就要考虑顾客不能买到饮料的情况。这也要分成两部分来考虑，一部分是顾客正好投入的是 5 角钱，不涉及找零钱的问题，这时只需要考虑两种饮料的数量，饮料卖完，后面的顾客将不能得到饮料，并拿回自己的钱；另外一种情况是顾客投入的是 1 元钱，这时不仅要考虑饮料是否存在，还要考虑是否有足够的零钱找给顾客，没有需要的饮料和没有零钱找给顾客都将导致顾客不能得到饮料并拿回自己的钱。下面分别测试这两种情况。

① 顾客只投入 5 角钱。针对这种情况就 beer 和 orange 分别设计了如下的两个测试类：testSaleMachineA_Error()(顾客不能用 5 角钱买到 beer)，testSaleMachineB_Error() (顾客不能用 5 角钱买到 orange)。鉴于 beer 和 orange 的数量被设为 3，于是我们的思路是：顾客前三次都可以买到自己需要的饮料，第四次则不能，系统提示顾客拿回自己的钱。

```
public void testSaleMachineA_Error() {
    SaleMachine salemachine=new SaleMachine();
```

```
        assertEquals("You have pay for the beer. Please pick it up.",
                    salemachine.Operate("beer",  5));
       assertEquals("You have pay for the beer. Please pick it up.",
                    salemachine.Operate("beer",  5));
       assertEquals("You have pay for the beer. Please pick it up.",
                    salemachine.Operate("beer",  5));
        assertEquals("There has no beer, please pick up your money, Sorry!",
                    salemachine.Operate("beer",  5));
}
```

同理，编写 testSaleMachineB_Error()：

```
public void testSaleMachineB_Error() {
        SaleMachine salemachine=new SaleMachine();
        assertEquals("You have pay for the orange. Please pick it up.",
                    salemachine.Operate("orange",  5));
        assertEquals("You have pay for the orange. Please pick it up.",
                    salemachine.Operate("orange",  5));
        assertEquals("You have pay for the orange. Please pick it up.",
                    salemachine.Operate("orange",  5));
        assertEquals("There has no orange, please pick up your money, Sorry!!",
                    salemachine.Operate("orange",  5));
}
```

这时的测试结果如图 6-13 所示。

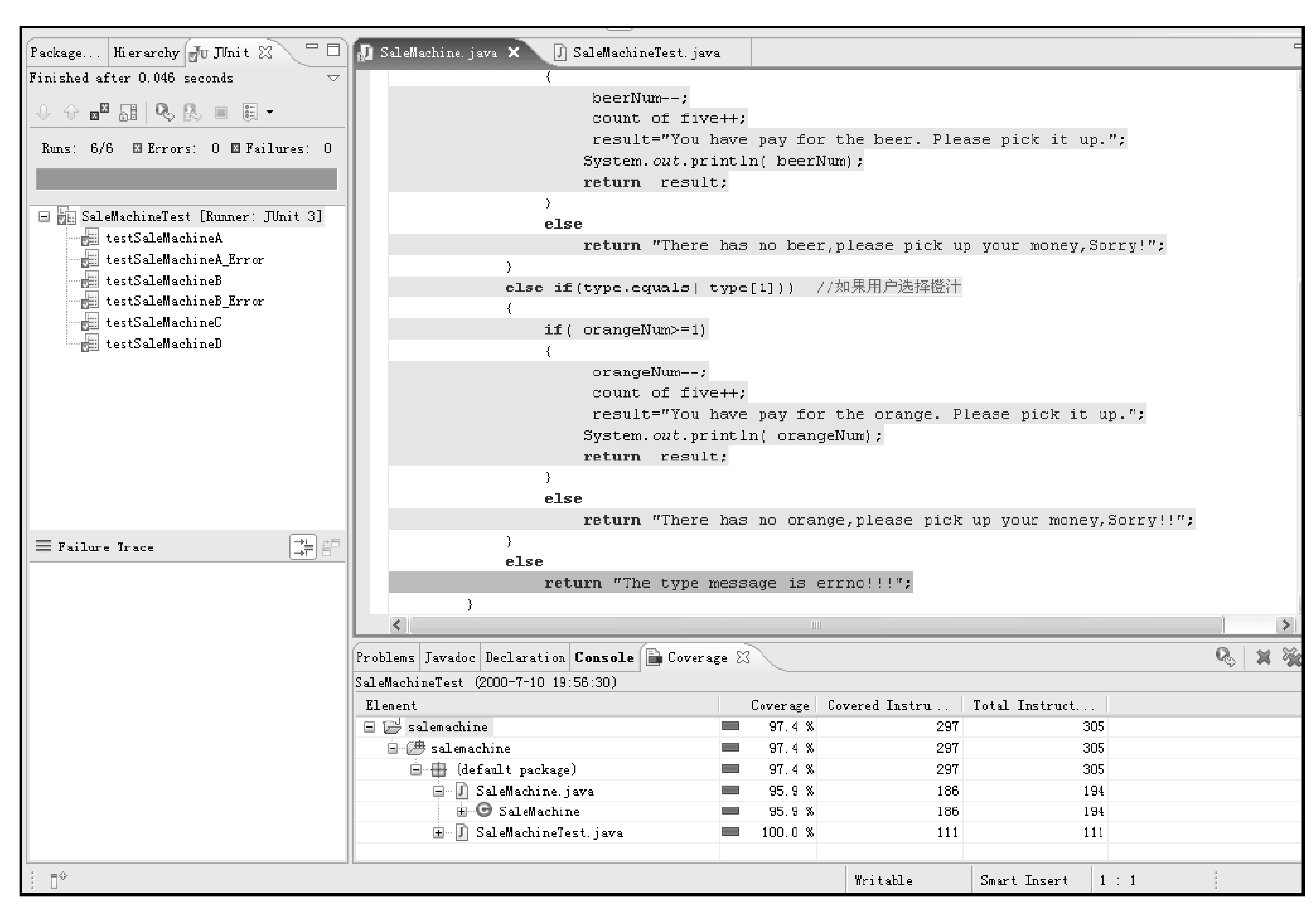


图 6-13　补充顾客用 5 角钱不能买到饮料的测试用例后的覆盖测试结果

从图 6-13 可以看出，已经设计的 6 个测试类全部通过，且全部覆盖，从右下视图框还可以看出测试覆盖率提高到了 97.4%。

图 6-14 显示了还需补充的测试用例警告(参见彩图)。

```
SaleMachine.java ×    SaleMachineTest.java
            else
                return "The type message is errno!!!";
        }
        else if(money==1)    //如果用户投入一元钱
        {
            if(type.equals( type[0])&& beerNum>=1)   //如果用户选择啤酒且还有啤酒
            {
                if( count of five>=1)    //如果有零钱找
                {
                    beerNum--;
                    count of five--;
                    count of one++;
                    result="You have pay for the beer. Please pick it
                    return  result;
                }
                else
                    return "There has no loose change,Please pick up your money,Sorry!!!!";
            }
            else if(type.equals( type[1])&& orangeNum>=1)    //如果用户选择橙汁且还有橙汁
            {
                if( count of five>=1)    //如果有零钱找
                {
                    orangeNum--;
                    count of five--;
                    count of one++;
                    result="You have pay for the orange. Please pick it up and the loose cha
                    return  result;
                }
                else
                    return "There has no loose change,Please pick up your money,Sorry!!!!!";
            }
        }
        return "There has some input error!!!!!";
    }
```


图 6-14　还需补充的测试用例警告

② 然后考虑顾客投入 1 元钱时的情况。这个时候要保证有饮料、有零钱可以找回的情况下顾客才能得到饮料。因此，有无饮料、有无零钱是必须要考虑的两个因素，缺一不可。缺少任一条件，都将使顾客无法得到饮料。因此设计了 testSaleMachineC_Error()(顾客投了 1 元钱但是不能买到 orange)和 testSaleMachineD_Error()(顾客投了 1 元钱但是不能买到 beer)这两个测试类。

```
public void testSaleMachineC_Error() {
    SaleMachine salemachine=new SaleMachine();
    assertEquals("You have pay for the orange. Please pick it up and the loose change.",
            salemachine.Operate("orange",   1));
    assertEquals("You have pay for the orange. Please pick it up and the loose change.",
            salemachine.Operate("orange",   1));
    assertEquals("You have pay for the orange. Please pick it up and the loose change.",
            salemachine.Operate("orange",   1));
    assertEquals("There has no loose change，Please pick up your money，Sorry!!!!",
            salemachine.Operate("beer",   1));
}
```

同理，设计顾客投入 1 元不能买到 orange 的情况：

```
public void testSaleMachineD_Error() {
    SaleMachine salemachine=new SaleMachine();
    assertEquals("You have pay for the beer. Please pick it up and the loose change.",
            salemachine.Operate("beer", 1));
    assertEquals("You have pay for the beer. Please pick it up and the loose change.",
            salemachine.Operate("beer", 1));
    assertEquals("You have pay for the beer. Please pick it up and the loose change.",
            salemachine.Operate("beer", 1));
    assertEquals("There has no loose change, Please pick up your money, Sorry!!!!!",
            salemachine.Operate("orange", 1));
}
```

将上述两个测试类加入到测试程序中，重新进行覆盖测试，可以得到如图 6-15 所示的源程序颜色变化情况。

由图 6-15 可以看到，不仅单元测试通过了，而且覆盖测试也顺利通过了，覆盖率达到了 98.9%。但是还是有没有覆盖到的地方，我们要达到的覆盖率目标是 100%，这就说明还需要继续设计测试类，以便覆盖测试能够真正地做到整个 Java 程序的所有语句和分支都被覆盖测试一遍。

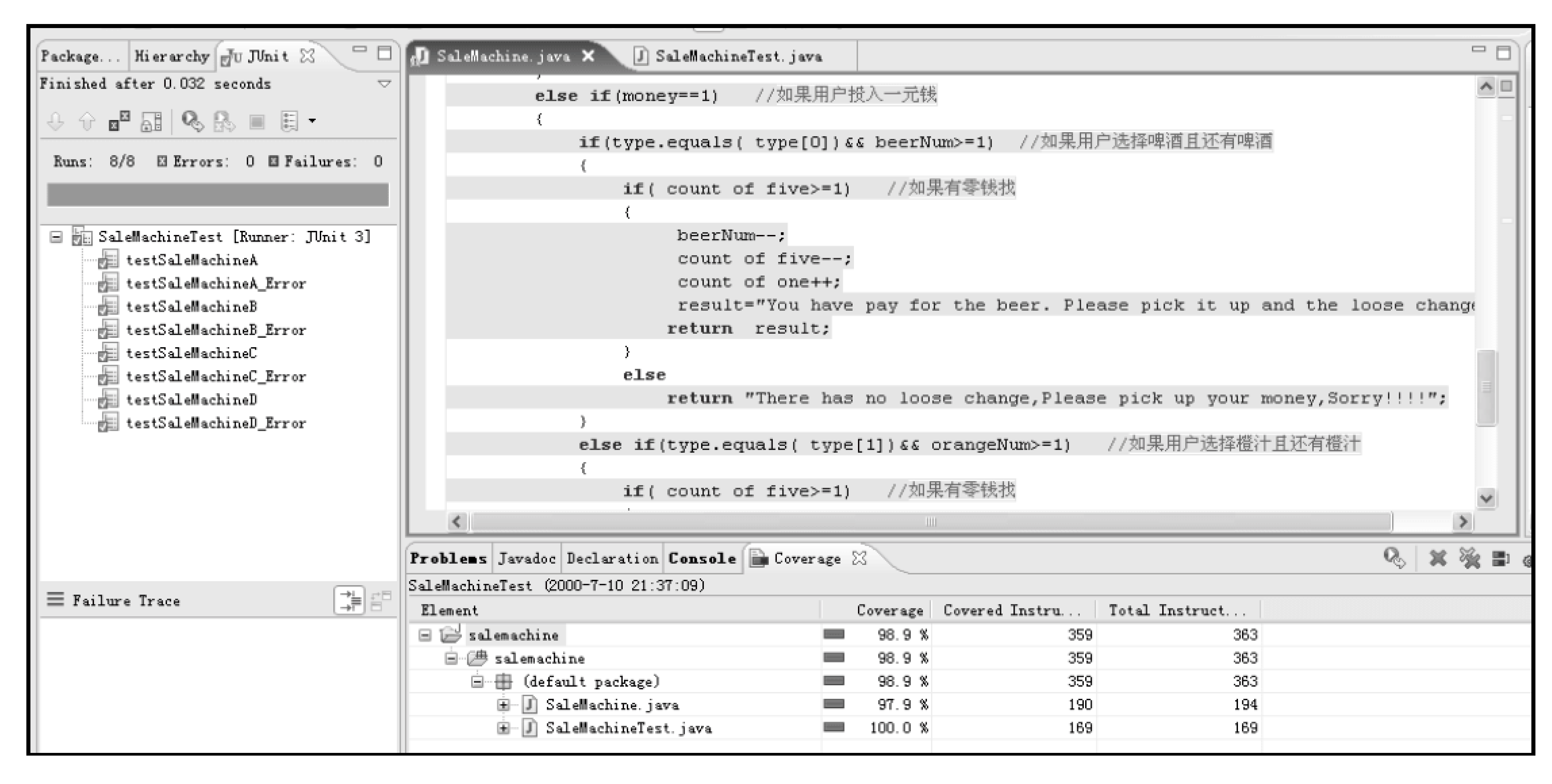


图 6-15　补充顾客用 1 元不能够买到饮料的测试用例后覆盖测试结果

③ 来看一下仅剩下的两条红色语句，那就是当顾客放进钱便直接报错，也就是说顾客提出的要求自动售货机根本没有实现的可能。一种情况是：顾客放进的是 5 角钱或是 1 元钱，但是他却想喝别的饮料，这是不可能的，因为售货机里根本没有这种饮料。另一种情况是：顾客要的的确是 beer 或是 orange 的其中一个，但是却投进了 100 元，售货机根本没有找到足够零钱的可能，这时候也是直接报错。针对这两种情况，可以设计 testSaleMachineError() (顾客投放 5 角钱或是 1 元钱，但要的却是别的饮料)和 testSaleMachine_FinalError()(顾客要的

是 beer 或是 orange，但是投放了 5 角钱和 1 元钱之外的其他面值的货币)这两个测试类，以保证覆盖测试可以涉及最后剩下的两条红色语句。

```
public void testSaleMachineError() {
    SaleMachine salemachine=new SaleMachine();
    assertEquals("The type message is errno!!!",
                salemachine.Operate("coca",  5));
 }
public void testSaleMachine_FinalError() {
    SaleMachine salemachine=new SaleMachine();
    assertEquals("There has some input error!!!!!",
                salemachine.Operate("beer",  100));
    assertEquals("There has some input error!!!!!",
                salemachine.Operate("orange",  100));
}
```

将这两个测试类添加到测试程序当中，来看一下最后的测试结果：全变绿了，覆盖率已经到达了 100%，如图 6-16 所示，这说明测试覆盖到了源程序的所有语句。

Problems　Javadoc　Declaration　Console　Coverage

SaleMachineTest (2000-7-10 21:54:27)

Element	Coverage	Covered Instru...	Total Instruct...
salemachine	100.0 %	391	391
salemachine	100.0 %	391	391
(default package)	100.0 %	391	391
SaleMachine.java	100.0 %	194	194
SaleMachine	100.0 %	194	194
SaleMachine()	100.0 %	17	17
init()	100.0 %	13	13
Operate(String, int)	100.0 %	164	164
SaleMachineTest.java	100.0 %	197	197
SaleMachineTest	100.0 %	197	197
setUp()	100.0 %	3	3
tearDown()	100.0 %	3	3
testSaleMachine_FinalError(	100.0 %	17	17
testSaleMachineA()	100.0 %	11	11
testSaleMachineA_Error()	100.0 %	29	29
testSaleMachineB()	100.0 %	11	11
testSaleMachineB_Error()	100.0 %	29	29
testSaleMachineC()	100.0 %	11	11
testSaleMachineC_Error()	100.0 %	29	29
testSaleMachineD()	100.0 %	11	11
testSaleMachineD_Error()	100.0 %	29	29
testSaleMachineError()	100.0 %	11	11

图 6-16　覆盖率达到 100%

用 EclEmma 仅能进行语句覆盖和分支覆盖等比较初级的覆盖测试，对于高级别的覆盖项目就要用到商用“白盒”测试工具了。尽管如此，EclEmma 能够很好地体现覆盖测试的思想，帮助读者掌握覆盖测试的知识和技能。

6.3　GCC 的覆盖测试工具 Gcov

Gcov 是 GNU/GCC 的工具组件，它可以作为 C/C++代码覆盖率的测试工具，使用起来非常便捷，不需要再进行配置，只需要准备好待测程序即可。但是它不是可视化界面，如

果想使用可视化界面，可使用 Gcov 等。

6.3.1　Gcov 测试环境建立

由于 Gcov 是内嵌于 GCC 的内部工作组件，所以不需要再进行配置，只需要准备好被测程序即可。可以用 Eclipse 作为测试平台(3.3 节介绍过如何建立测试环境)，也可以直接利用 Linux 提供的 GCC 编译器。

6.3.2　Gcov 测试功能及使用流程

Gcov 的基本功能是通过该工具可以查看测试时代码执行的覆盖率，支持函数覆盖、语句覆盖和分支覆盖等覆盖测试内容，帮助我们分析被测程序中的缺陷。使用该工具还可以查看程序在某分支处的执行频率，进而分析程序的性能。

Gcov 必须和 GCC 编译器结合使用，在编译时必须加上“-ftest-coverage -fprofile-arcs”选项，然后生成“./a.out”和“<源文件名>.gcda”这两个文件。Gcov 还有很多参数可以选择，详见图 6-17。

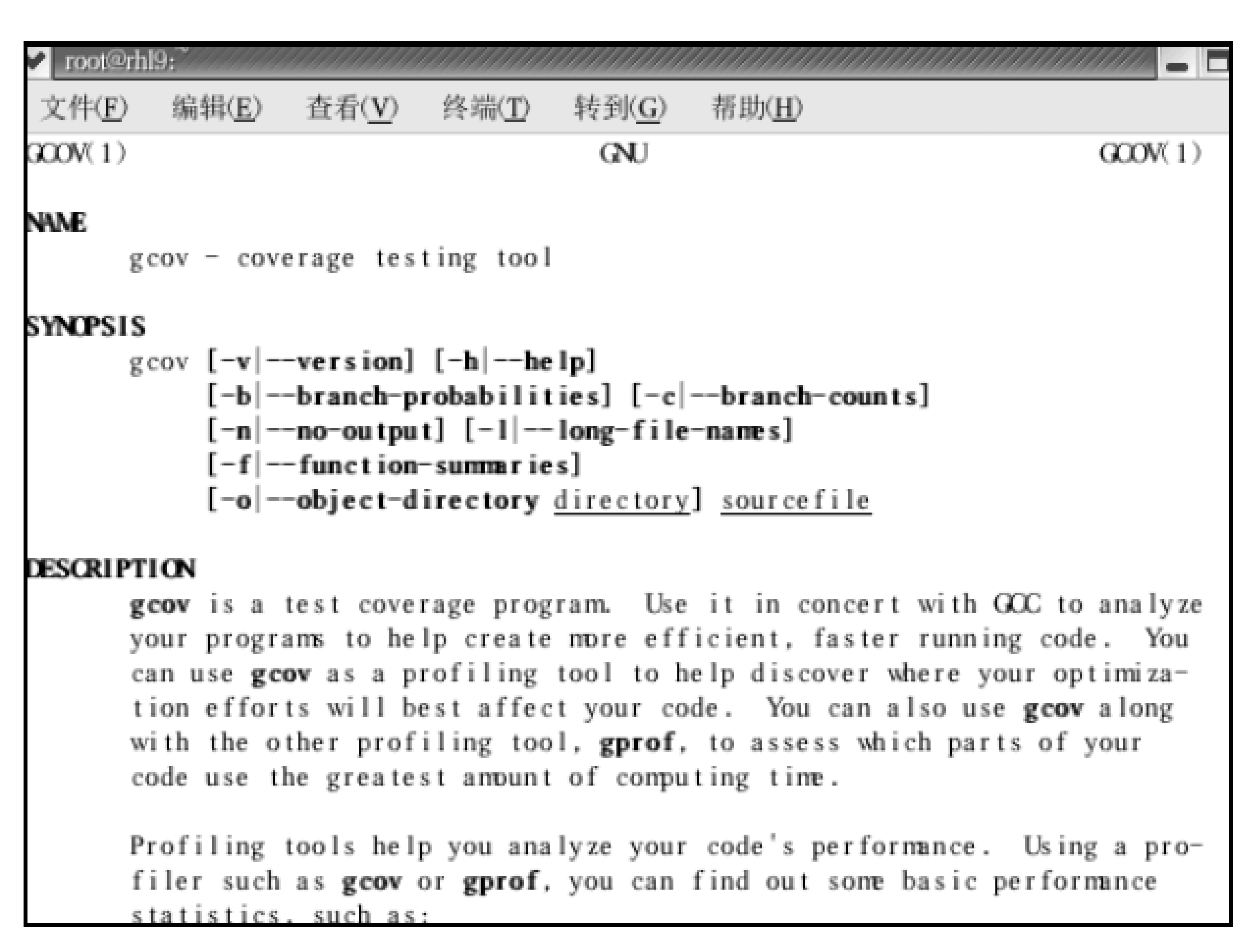


图 6-17　Gcov 参数列表

执行“./a.out”，根据源文件的语句、条件、路径、边界等因素，设计好需要输入的测试用例数据，这将自动记录在 Gcov 里面，显示覆盖率，然后生成一个名为“<源文件名>.c.gcov”的文件，里面显示了每条语句的执行次数，没有执行的语句则以“#####”标注。

在使用 Gcov 时，需注意该工具每次重新编译后，统计数据会被清空。所以项目测试必须在确定的版本上进行，若版本有修正，则应使用 Gcov 工具对变更部分进行补充测试。

可以根据一个小例子来具体解释 Gcov 的工作流程。代码如下：

```
#include <stdio.h>
int main(){
printf("hello，world");
```

```
        return 0;
}
```

利用 GCC 编译器对其进行编译：

```
cmd>gcc test.c -o test -ftest-coverage -fprofile-arcs
```

执行编译生成的可执行文件：

```
cmd>test.exe
```

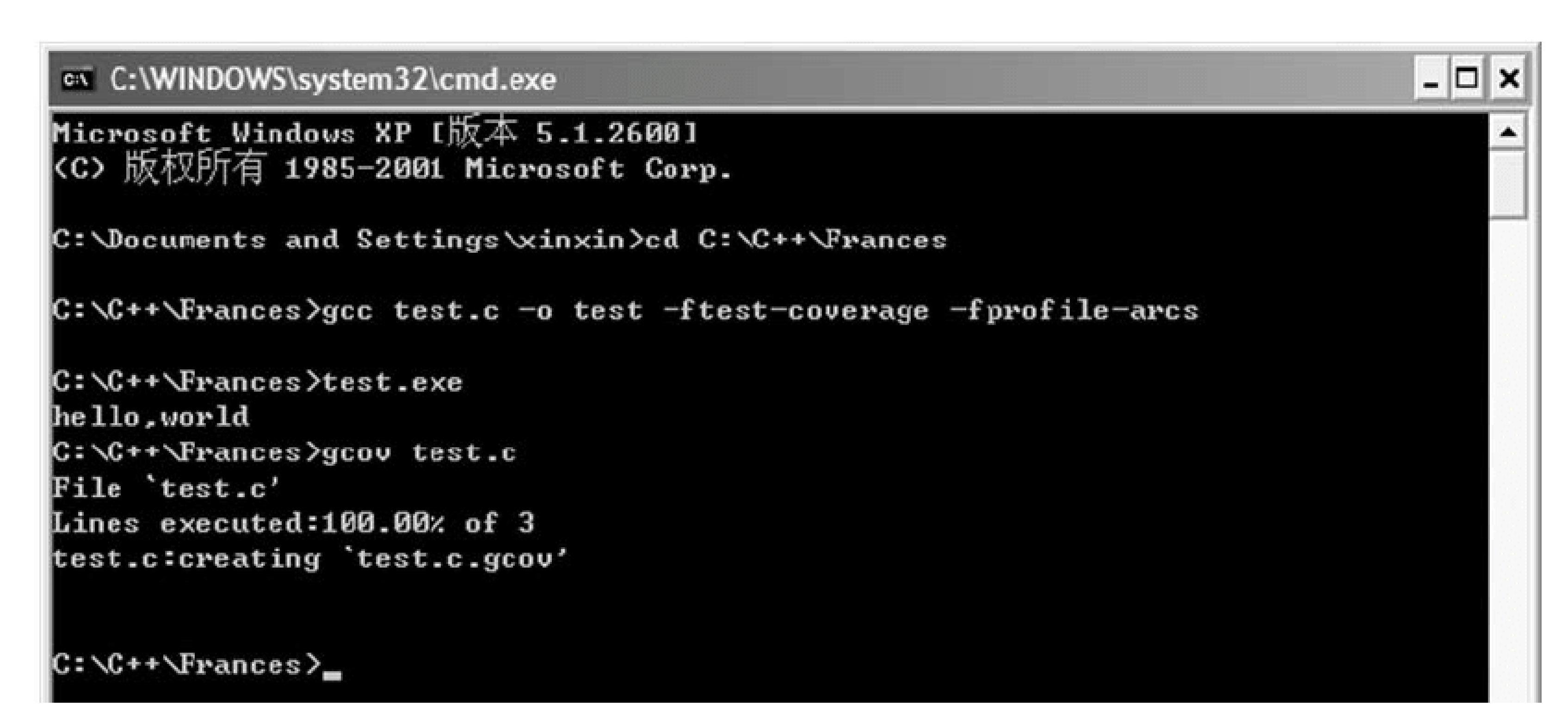
当运行可执行程序时，会生成一些包含关于程序的相关数据的文件。Gcov 程序将会使用这些文件来报告数据，并向开发者提供相应的信息。

当指定-ftest-coverage 选项时，会为每一个源码生成两个文件；这些文件以.bb 和.bbg 作为扩展名，可以用这些文件来重组每一个可执行程序的程序流图。对于-fprofile-arcs 选项来说，将会生成一个包含每一个指令分支的执行计数的以.da 为扩展名的文件；这些文件会在执行以后与源码文件一起使用，来标识源码的执行行为。

可以使用 Gcov 来生成代码覆盖率信息：

```
cmd>gcov test.c
```

可以看到覆盖率达到了 100%，如图 6-18 所示。可以通过查看所生成的 test.c.gcov 文件来了解每一源码行所实际运行的次数。



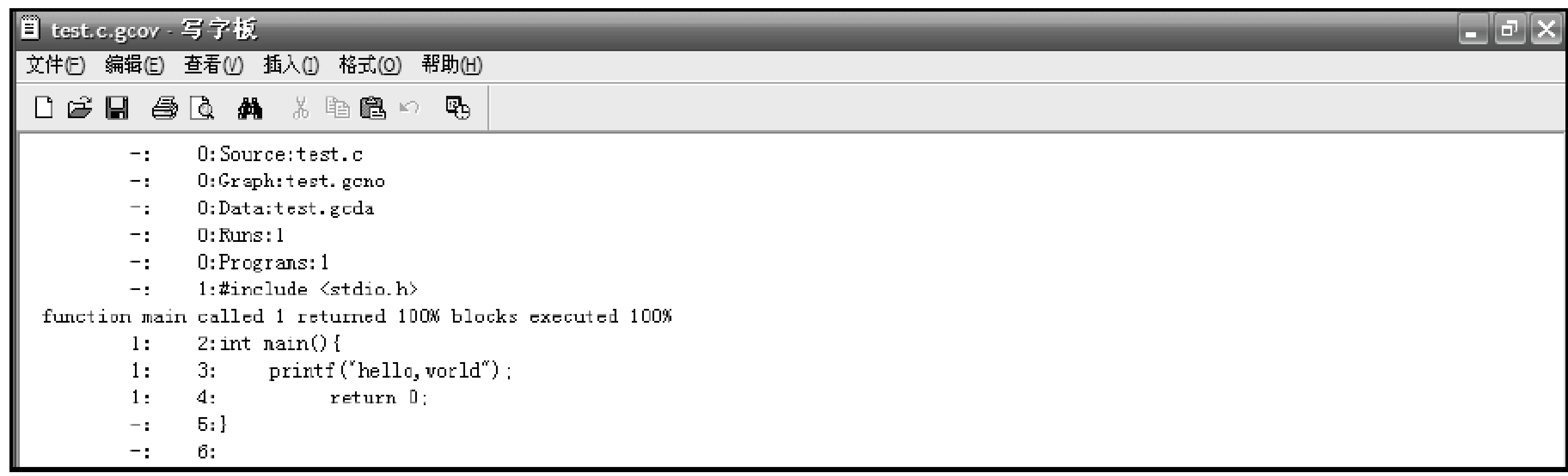
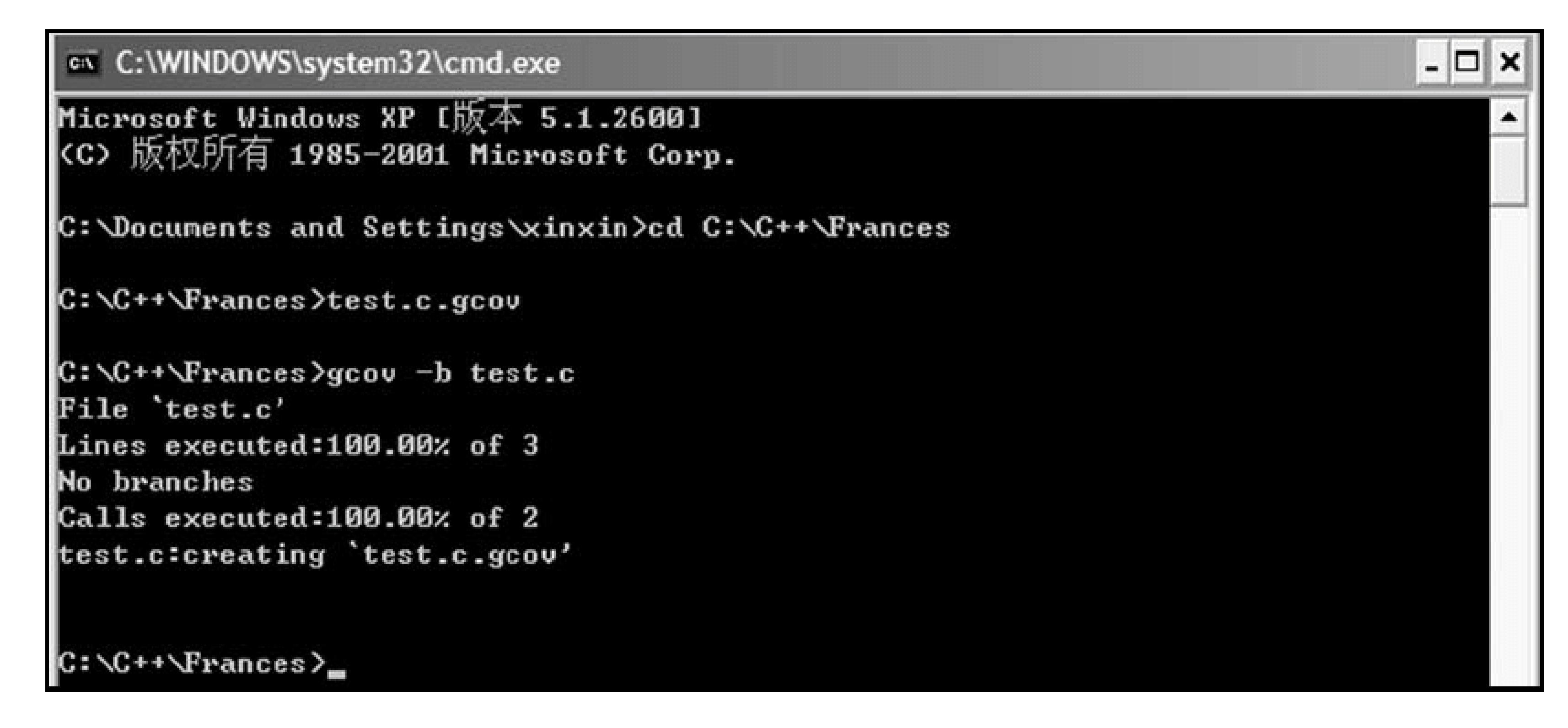


图 6-18　语句覆盖测试结果

当 Gcov 计数一个测试并不是 100%的覆盖时，将没有执行的行标记为“####”，而不是执行次数。此外，还可以查看分支执行频率：可以使用-b 选项来查看程序的分支数据。

这个选项会输出程序中每一个分支的频度与相应的摘要，所生成的 test.c.gcov 文件如图 6-19 所示。



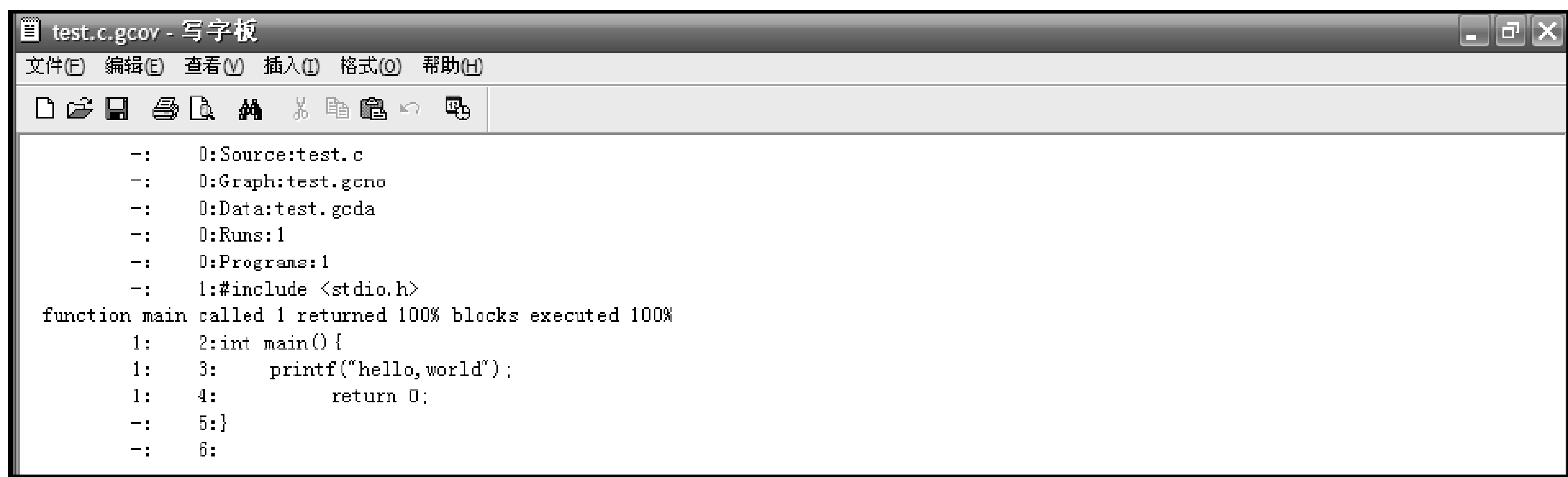


图 6-19　语句、分支覆盖测试结果

查看函数的执行情况，cmd>可以使用“-f”选项来查看每一个函数的执行情况。

6.3.3　Gcov 覆盖测试应用举例

下面同样以前面提到的售货机程序为例。

1. 被测程序代码

```
//autosell.c
#include<stdio.h>
void welcome(void);
void nochange(int num5coins);
void getcoin(int* coin);
void pushbutton(int* button);
void process(int* coin， int* button， int* num5coins);
int main(){
   int coin=0;
   int button=0;
   int num5coins=2;
   int i;
   for(i=0;i<10;i++){
```

```
        welcome();
        nochange(num5coins);
        getcoin(&coin);
        pushbutton(&button);
        process(&coin， &button， &num5coins);
    }
    return 0;
}
void welcome(){
    //clrscr();
    printf("Welcome to this auto selling machine!\n\n");
}
void nochange(int num5coins){
    if(num5coins= =0)
        printf("No Change Now!\n\n");
}
void getcoin(int* coin){
    int flagredo;
    do{
        printf("Please pitch your coin(5 for 5 jiao， 10 for 1 yuan):");
        scanf("%d"， coin);
        if(*coin!=5 && *coin!=10){
            printf("Wrong coin! Return the coin.\n\n");
            flagredo=1;
        }
        else
            flagredo=0;
    }while(flagredo);
}
void pushbutton(int* button){
    int flagredo;
    do{
        printf("Please select your drink(1 for orange juice， 2 for beer):");
        scanf("%d"， button);
        if(*button!=1 && *button!=2){
            printf("Wrong input，  please re-select.\n\n");
            flagredo=1;
        }
        else
            flagredo=0;
    }while(flagredo);
```

```
}
void process(int* coin， int* button，  int* num5coins){
    if(*coin= =10 && *num5coins= =0){
        printf("No change!\n");
        printf("Return 1 yuan coin.\n");
    }
    else{
        if(*coin= =10){
            if(*button= =1)
                printf("Please take your orange juice.\n");
            else
              printf("Please take your beer.\n");
            (*num5coins)– –;
            printf("Return 5 jiao coin.\n");
        }
        if(*coin= =5){
            if(*button= =1)
              printf("Please take your orange juice.\n");
            else
              printf("Please take your beer.\n");
            (*num5coins)++;
        }
    }
    printf("\nPress ENTER to continue");
    getchar();getchar();
    *coin=0;
    *button=0;
}
```

2. 终端输入

```
[root@rhl9 root]# gcc -fprofile-arcs -ftest-coverage autosell.c
[root@rhl9 root]# ./a.out
```

3. 输入设计好的测试用例

```
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao， 10 for 1 yuan):5
Please select your drink(1 for orange juice， 2 for beer):1
Please take your orange juice.

Press ENTER to continue
```

```
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):5
Please select your drink(1 for orange juice，2 for beer):2
Please take your beer.

Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):10
Please select your drink(1 for orange juice，2 for beer):1
Please take your orange juice.
Return 5 jiao coin.

Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):10
Please select your drink(1 for orange juice，2 for beer):2
Please take your beer.
Return 5 jiao coin.

Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):5
Please select your drink(1 for orange juice，2 for beer):0
Wrong input，  please re-select.

Please select your drink(1 for orange juice，2 for beer):1
Please take your orange juice.

Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):5
Please select your drink(1 for orange juice，2 for beer):0
Wrong input，  please re-select.

Please select your drink(1 for orange juice，2 for beer):2
Please take your beer.
```

```
Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):0
Wrong coin! Return the coin.

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):5
Please select your drink(1 for orange juice，2 for beer):0
Wrong input，  please re-select.

Please select your drink(1 for orange juice，2 for beer):1
Please take your orange juice.

Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):0
Wrong coin! Return the coin.

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):10
Please select your drink(1 for orange juice，2 for beer):2
Please take your beer.
Return 5 jiao coin.

Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):10
Please select your drink(1 for orange juice，2 for beer):0
Wrong input，  please re-select.

Please select your drink(1 for orange juice，2 for beer):1
Please take your orange juice.
Return 5 jiao coin.

Press ENTER to continue
Welcome to this auto selling machine!

Please pitch your coin(5 for 5 jiao，10 for 1 yuan):5
Please select your drink(1 for orange juice，2 for beer):5
```

```
Wrong input，  please re-select.

Please select your drink(1 for orange juice，2 for beer):3
Wrong input，  please re-select.

Please select your drink(1 for orange juice，2 for beer):2
Please take your beer.

Press ENTER to continue
```

4. 测试结果

1) 显示总体覆盖率

```
[root@rhl9 root]# gcov autosell.c
 95.74% of 56 source lines executed in file autosell.c
Creating autosell.c.gcov.
```

2) 显示函数覆盖率

```
[root@rhl9 root]# gcov -f autosell.c
100.00% of 12 source lines executed in function main
100.00% of 2 source lines executed in function welcome
 66.67% of 3 source lines executed in function nochange
100.00% of 10 source lines executed in function getcoin
100.00% of 10 source lines executed in function pushbutton
 89.47% of 19 source lines executed in function process
 94.64% of 56 source lines executed in file autosell.c
Creating autosell.c.gcov.
```

3) 显示分支覆盖率

```
[root@rhl9 root]# gcov -b autosell.c
 94.64% of 56 source lines executed in file autosell.c
 95.24% of 21 branches executed in file autosell.c
 95.24% of 21 branches taken at least once in file autosell.c
 86.96% of 23 calls executed in file autosell.c
Creating autosell.c.gcov.
```

5. 生成文件

```
//autosell.c.gcov
        #include<stdio.h>

        void welcome(void);
        void nochange(int num5coins);
```

```
        void getcoin(int* coin);
        void pushbutton(int* button);
        void process(int* coin， int* button， int* num5coins);

              2      int main(){
              2         int coin=0;
              2         int button=0;
              2         int num5coins=2;
              2         int i;
             22         for(i=0;i<10;i++){
branch 0 taken = 91%
branch 1 taken = 100%
branch 2 taken = 100%
             20            welcome();
call 0 returns = 100%
             20            nochange(num5coins);
call 0 returns = 100%
             20            getcoin(&coin);
call 0 returns = 100%
             20            pushbutton(&button);
call 0 returns = 100%
             20            process(&coin， &button， &num5coins);
call 0 returns = 100%
           }
              2         return 0;
        }

             20      void welcome(){
         //clrscr();
             20         printf("Welcome to this auto selling machine!\n\n");
call 0 returns = 100%
        }
             20      void nochange(int num5coins){
             20         if(num5coins= =0)
branch 0 taken = 100%
         ######            printf("No Change Now!\n\n");
call 0 never executed
        }
             20      void getcoin(int* coin){
             20         int flagredo;
             26         do{
```

```
                26              printf("Please pitch your coin(5 for 5 jiao，10 for 1 yuan):");
call 0 returns = 100%
                26              scanf("%d"，coin);
call 0 returns = 100%
                26              if(*coin!=5 && *coin!=10){
branch 0 taken = 46%
branch 1 taken = 57%
                 6                  printf("Wrong coin! Return the coin.\n\n");
call 0 returns = 100%
                 6                  flagredo=1;
branch 0 taken = 100%
               }
               else
                20                  flagredo=0;
                26          }while(flagredo);
branch 0 taken = 23%
        }
                20      void pushbutton(int* button){
                20          int flagredo;
                29          do{
                29              printf("Please select your drink(1 for orange juice，2 for beer):");
call 0 returns = 100%
                29              scanf("%d"，button);
call 0 returns = 100%
                29              if(*button!=1 && *button!=2){
branch 0 taken = 31%
branch 1 taken = 55%
                 9                  printf("Wrong input，  please re-select.\n\n");
call 0 returns = 100%
                 9                  flagredo=1;
branch 0 taken = 100%
               }
               else
                20                  flagredo=0;
                29          }while(flagredo);
branch 0 taken = 31%
        }
                20      void process(int* coin，int* button，  int* num5coins){
                20          if(*coin= =10 && *num5coins= =0){
branch 0 taken = 60%
branch 1 taken = 100%
```

```
        ######          printf("No change!\n");
call 0 never executed
        ######          printf("Return 1 yuan coin.\n");
call 0 never executed
branch 1 never executed
            }
            else{
                20          if(*coin= =10){
branch 0 taken = 60%
                8               if(*button= =1)
branch 0 taken = 63%
                3           printf("Please take your orange juice.\n");
call 0 returns = 100%
branch 1 taken = 100%
                    else
                5           printf("Please take your beer.\n");
call 0 returns = 100%
                8               (*num5coins)– –;
                8               printf("Return 5 jiao coin.\n");
call 0 returns = 100%
                }
                20          if(*coin= =5){
branch 0 taken = 40%
                12              if(*button= =1)
branch 0 taken = 50%
                6           printf("Please take your orange juice.\n");
call 0 returns = 100%
branch 1 taken = 100%
                    else
                6           printf("Please take your beer.\n");
call 0 returns = 100%
                12              (*num5coins)++;
                }
            }
                20      printf("\nPress ENTER to continue");
call 0 returns = 100%
                20      getchar();getchar();
call 0 returns = 100%
call 1 returns = 100%
                20      *coin=0;
                20      *button=0;
```

```
}
```

覆盖测试能够与开发交织共同进行，以便在开发过程中能更快地验证测试的质量，发现 Bug 并进行改正。好的测试用例能够充分地测试软件，能够更快更早地发现 Bug，节约成本和时间。优秀的覆盖测试不能证明软件中不存在 Bug，但可以证明软件的质量是优秀的，树立对软件的信心。覆盖测试最好的测试员是开发参与者或编码者，只有这样才能更好地保证测试的有效性与效率。覆盖测试常与单元测试结合使用，方便、高效、节约。

实 验 习 题

1. 基于 nUnit 对.NET 程序实施覆盖测试，设计出各种测试用例，完成 100%的覆盖。

2. 尝试应用 COVTOOL(如果 GCC 3 及以上版本有问题，可选择 GCC 2.9.x)对 C/C++程序进行测试，完成覆盖率 100%的要求，并与 CppUnit 进行比较。

第Ⅳ部分　图形用户界面测试篇

界面是软件与用户交互的最直接的层面，界面的好坏决定了用户对软件的第一印象。目前，广为流行和使用的是图形用户界面(GUI)，设计良好的 GUI 能够引导用户自己完成相应的操作，起到向导的作用。同时 GUI 如同人的面孔，具有吸引用户的直接优势。设计合理的 GUI 能给用户带来轻松愉悦的感受和成功的感觉，相反由于 GUI 设计得不好，常常会给用户带来使用上的不便，影响产品的推广和普及。目前软件设计人员对 GUI 设计的重视程度还远远不够，直到近些年 Web 应用及网页制作的兴起，GUI 设计才引起各方的关注和重视。

目前流行的界面风格有三种：多窗体风格、单窗体风格以及资源管理器风格。无论哪种风格，以下规则都是应该被重视的。

(1) 易用性：按钮名称应该易懂，用词准确，摒弃有二义性的字眼，要与同一界面上的其他按钮易于区分。理想的情况是，用户不用查阅帮助就能知道该界面的功能，并进行相关的正确操作。

(2) 规范性：通常界面都按 Windows 界面的规范来设计，可以说，界面遵循规范化的程度越高，其易用性就会越好。小型软件一般不提供工具箱。

(3) 帮助设施：系统应该提供详尽而可靠的帮助文档，使用户在对使用产生迷惑时可以自己寻求解决方法。

(4) 合理性：屏幕对角线相交的位置是用户直视的地方，正上方四分之一处为易吸引用户注意力的位置，在放置窗体时要注意利用这两个位置。

(5) 美观与协调性：界面大小应该符合美学观点，感觉协调舒适，能在有效的范围内吸引用户的注意力。

(6) 菜单位置：菜单是界面上最重要的元素，菜单位置应按照功能来组织。

(7) 独特性：如果一味地遵循业界的界面标准，则会丧失自己的个性。在框架符合以上规范的情况下，设计具有自己独特风格的界面尤为重要。尤其是在商业软件流通中，这会起到很好的潜移默化的广告效用。

(8) 快捷方式的组合：在菜单及按钮中使用快捷键可以让喜欢使用键盘的用户操作得更快一些。在西文 Windows 及其应用软件中，快捷键的使用大多是一致的。这些快捷键也可以作为开发中文应用软件的标准，当然也可以使用汉语拼音的开头字母。

(9) 安全性考虑：在界面上通过下列方式来控制出错几率，可大大减少系统因用户人为错误而引起的破坏。开发者应当尽量周全地考虑到各种可能发生的问题，使出错的可能降至最小。如应用出现保护性错误而导致退出系统，这种错误最容易使用户对软件失去信

心。因为这意味着用户要中断思路，并费时费力地重新登录，而且已进行的操作也会因没有存盘而全部丢失。

(10) 多窗口的应用与系统资源：设计良好的软件不仅要有完备的功能，而且要尽可能地占用最低限度的资源。

目前的 GUI 应用最广泛的是桌面软件、C/S 应用的客户端软件以及基于 Web 应用的 B/S 浏览器端软件。事实上，GUI 测试属“黑盒”测试或功能测试。GUI 测试和评估的重点是正确性、易用性和视觉效果，GUI 中的文字检查和拼写检查也是 GUI 测试的重要环节。尽管 GUI 有时依赖于测试人员的主观判断，但 GUI 测试也要遵循一些基本原则，如易用性、规范性、合理性、美观与协调性、菜单位置、独特性、快捷方式的组合、排错性考虑等。

用户通过 GUI 拖动鼠标、单击按钮等来驱动软件完成需要的功能。这里面涉及：

(1) GUI 输入主要是由一系列事件或行动组成，拥有大量的需要测试的状态，输入和状态转移不仅与当前输入事件相关，而且与以前的输入事件相关。

(2) GUI 软件的输入和事件的排列数目非常巨大，GUI 软件拥有大量的执行路径。

(3) GUI 中包含大量复杂的组件或控件，这里面往往允许多个窗口被激活，彼此之间存在一些同步机制，而且一个窗口中的每个对象可以影响或控制窗口中的其他对象。

(4) GUI 软件的各个界面之间互相依赖，互相影响。

(5) GUI 软件的执行是基于事件和消息的，可以通过鼠标事件或键盘事件驱动 GUI 软件的执行，也可以通过发送系统内部消息(如定时器消息)驱动 GUI 软件的执行，具有事先不确定性。

(6) GUI 测试中的许多测试方法是重复而枯燥的，如有的地方需要在 GUI 上单击按钮上百次等，诸如此类的情况还有很多，这对于测试人员来说既枯燥又需花费很多时间。

(7) 除了要考虑 GUI 图形对象测试本身的特点外，对系统工作流程的测试也是一个重点。因为数据库设计、程序结构设计最终是要反映在界面设计上的。

事实上，GUI 的测试过程是执行用户操作步骤、获取检测状态和对比验证内容。从上面可以看到，由于 GUI 的诸多特点以及用户操作的随意性，可能会出现无限种事件交互方式，所以必须考虑各种复杂事件的情况，以及各种验证状态和内容。为提高效率，必须避免极耗资源的手工测试，而采用 GUI 测试工具或自动化测试工具。

GUI 自动化测试工具在软件测试时的基本原理是：在测试者运行应用程序的同时，把他的所有动作(包括键盘操作、鼠标操作等)捕获或录制下来，生成一个脚本文件，这个脚本以后可以被“回放”(playback)，也就是按照上一次的所有执行动作重复执行一遍，实现自动运行和测试。在实际的测试过程中，脚本通常按同一动作重复执行的意义并不大，而是要根据测试需求对其进行一些必要的修改。

GUI 自动化测试工具的捕获/录制模式只能帮助测试人员获得测试用例的原型级的实现。为了插入检查点或者对脚本进行维护，大多数录制的脚本都需要进行编程修改。因此，GUI 测试自动化也是一种编程的任务。另一方面，它还是一种软件规范的过程。在录制之前，应该设计好测试用例，它们的规范要比手工测试详细得多。

目前 GUI 自动化测试工具有很多，商用的最知名的 GUI 测试工具有：HP 的 UFT、IBM Rational 的 Robot、Compuware 公司的 QARun 等。开源的有：Linux 下的 GUI 测试工具 Xnee，测试 Java 的 AWT、Swing、SWT 界面的 Selenium，用 Java/Swing 开发的用于 GUI 应用程序的测试框架 Marathon，Flex 自动化 GUI 测试工具 RIATest 等。

下面介绍一个开源的 GUI 自动化测试工具——SWTBot。SWTBot 是一个用于 SWT、基于 Eclipse 应用的 GUI 测试工具，提供了能简化访问 SWT 和 Eclipse 组件的 API，而且 SWTBot 可以运行于在所有平台上运行的 SWT。测试脚本可以通过 Ant 任务运行，因此可以把测试作为构件集成进来。SWTBot 基于 Apache 2 许可协议。

SWTBot 可以用来模拟用户鼠标的单击行为，可以在程序中预先设定鼠标的单击顺序，之后 SWTBot 就会按照设定的顺序进行操作。SWTBot 测试运行配置和 JUnit 非常相似，测试方法结构都差不多，实际上它继承自 JUnit 的方法，编写 SWTBot 测试代码的方式和 JUnit 一样。

本篇介绍的大多数工具均以 JUnit 为基础进行框架的拓展，因此前面章节对 JUnit 的学习是必需的，使用也必须是熟练的。

第7章　Java GUI基础类库应用测试

进行 Java GUI 编程，一般是在 SWT/JFace、Swing 和 AWT 之间选择。AWT 是 Java 语言的第一个 GUI 类库包；Swing 兼容 AWT，同时又对 AWT 进行了改进，是 Java 语言的第二个 GUI 类库包；SWT/JFace 则采取了与 AWT 和 Swing 完全不同的技术路线。

AWT(Abstract Windowing Toolkit，抽象窗口工具包)，是 Java 提供的用来建立和设置 Java 图形用户界面的基本工具。AWT 由 Java 中的 java.awt 包提供，里面包含许多可用来建立与平台无关的 GUI 的类，这些类被称为组件。

AWT 是 Java 的平台独立的窗口系统、图形和用户界面构件工具包。AWT 是 Java 基础类(JFC)的一部分，为 Java 程序提供 GUI 的标准 API。

AWT 提供了 Java Applet 和 Java Application 中可用的 GUI 中的基本组件。由于 Java 是一种独立于平台的程序设计语言，而 GUI 却往往是依赖于特定平台的；因而 Java 采用了相应的技术，使得 AWT 能提供给应用程序独立于机器平台的接口，这保证了同一程序的 GUI 在不同机器上运行具有类似的外观。

AWT 的 API 为 Java 程序提供了建立 GUI 的工具集，AWT 可用于 Java 的 Applet 和 Applications 中。它支持 GUI 编程的功能包括：用户界面组件；事件处理模型；图形和图像工具，包括形状、颜色和字体类；布局管理器，可以进行灵活的窗口布局而与特定窗口的尺寸和屏幕分辨率无关；数据传送类，可以通过本地平台的剪贴板来进行剪切和粘贴。

AWT 在设计上与 Java“一次编写，到处运行”的信条存在冲突。一个 AWT 应用可能在 Windows 上表现得很好，可是到了 Macintosh 上却几乎不能使用。因此，在第二版的 Java 开发包中，AWT 的器件在很大程度上被 Swing 工具包替代。Swing 通过自己绘制器件而避免了 AWT 的种种弊端。Swing 调用本地图形子系统中的底层例程，而不是依赖操作系统的高层用户界面模块。Swing 的出现，宣告了 AWT 的穷途末路，目前几乎看不到 AWT 在 GUI 上的应用了。

JFC（Java Foundation Classes）是一个图形框架，应用该框架可建构出可移植的 Java 图形用户界面 GOI。

Java Swing 是 JFC 的一部分，它解决了 AWT 的很多缺点。相对于 AWT，Swing 是轻量级元件。Swing 提供了许多比 AWT 更好的屏幕显示元素，它们用纯 Java 写成，所以同 Java 本身一样可以跨平台运行，这一点儿也不像 AWT。它们是 JFC 的一部分，支持可更

换的观感和主题(各种操作系统默认的特有主题)。然而 Swing 不是真地使用平台所提供的设备，而是仅仅在表面上模仿它们。这意味着可以在任意平台上使用 Java 支持的任意观感。轻量级元件的缺点是执行速度较慢，优点是可以在所有平台上采用统一的行为。

在 Swing 中，Sun 开发了一个经过仔细设计的、灵活而强大的 GUI 工具包。其中大量应用了 MVC 模式，这大大增加了 Swing 的灵活性。灵活就意味着功能强大，功能强大就意味着复杂，对于一般的程序员来说，Swing 太复杂了，以至于他们在还不了解 Swing 的时候就已经放弃了选择 Swing，或者失去静下心来继续学下去的毅力，最后写出来的只能是一堆垃圾代码。另外，Swing 的低效则让大多数的程序员感叹。由于 Swing 是轻量级元件，因此 Swing 中的每一个组件都是采用 Java 自身的画点、画线等函数画出来的，并没有调用操作系统组件。Java 字节码的运行速度大概是同等条件下 C/C++语言程序运行速度的 1/10～1/5。于是，采用 JBuilder 进行开发的程序员经常可以看到 JBuilder 灰屏(窗体上组件还没有画出来)的现象。正是因为 Swing 的蜗牛速度，导致在 Java 推出这么多年以来，很少能够看见比较成熟的 Swing 桌面应用(当然 JBuilder 算是其中最成功的一个了，但是现在随着 Eclipse 的崛起，JBuilder 的发展也是举步维艰)。

SWT/JFace 很好地解决了上述问题。虽然 Sun 不接纳 SWT/JFace 作为 Java 中的一种图形 API 标准，但它凭借着 Eclipse 的优异表现，以不可阻挡之势向前发展着。现在终于可以用 SWT/JFace 轻松地开发出高效率的 GUI 程序，且拥有标准的 Windows 外观。Eclipse 软件就是基于 SWT/JFace 构建的。

SWT/JFace 直接调用操作系统的图形库，从而使得 Java 应用程序的外观与操作系统的习惯完全一致。更为重要的是，SWT/JFace 采用有限调用本地方法(控件)，只有当本地找不到所需要的控件时，才进行模拟。对本地方法的直接调用大幅提高了基于 SWT/JFace 的 Java 应用程序的运行速度。SWT/JFace 具有比 AWT 更为丰富的控件，比 Swing 更为快捷的速度。SWT/JFace 的缺点主要在于两点：①不是 Java 语言标准；②某些平台对其不支持。

7.1　JFCUnit 单元测试工具介绍

前面章节介绍过 XP 测试驱动开发观念下的单元测试技术是令人瞩目的技术之一，其总体目标是负责软件在运行过程中的正确无误。在 Java 平台中，常常使用 JUnit 进行单元测试，它与 ANT 结合为软件的自动化单元测试的实现提供了一个理想的测试框架。鉴于 JUnit 的一些局限性，在它的框架基础之上，又延伸出适应于其他特殊情况下的测试框架，比如，适应于网络 Web 测试的 HttpUnit 和侧重于测试 GUI 的 JFCUnit 单元测试技术等。

现代软件中许多程序都要求有友好的图形界面，因而对 GUI 的测试就成为测试的一个重要部分。因为 GUI 图形界面测试是一种比较特殊的测试，它不完全是对逻辑的验证、对运算结果的判断，还包括很强的与用户的交互性。图形界面测试的被测目标除了包括单击按钮触发事件之类的客观内容外，还有很多是与用户的主观因素紧密结合的，如显示字体的大小、颜色是否美观、比例是否匀称等。鉴于 GUI 图形界面测试的特殊性，JUnit 已经

不能完全胜任，在此基础上 JFCUnit 应运而生。

实际上，JFCUnit 是目前流行的测试框架 JUnit 的扩展框架。JFCUnit 是在 JUnit 基础上针对 Swing GUI 扩展的单元测试工具。在同一个应用程序中，可以通过组件发现方法查找到组件，模拟用户动作触发组件事件来提供测试驱动，通过断言验证组件状态是否正确。JFCUnit 具有两大优点：继承了 JUnit，具有 JUnit 进行单元测试的所有优点；提供了一系列 GUI 组件发现方法及用户动作模拟方法。

为了使 GUI 测试简单化、自动化，JFCUnit 框架实现了对 JUnit 的延伸，弥补了 JUnit 在界面 GUI 测试中的不足；重点加强了对 GUI 的测试，使单元测试更加容易、全面。

JFCUnit 使程序员不仅能够实现 JUnit 的功能性单元测试，还能够为基于 Java Swing (AWT)的图形程序编写测试用例。除了包含 JUnit 功能外，它还提供如下基本功能：①获取窗体/对话框的句柄；②在一个继承的组件容器内，确定所需要的组件；③模拟触发组件的事件；④用于 GUI 测试的多线程方法。

JFCUnit 从 2.0 版本开始，提供 XML 录制和回放功能。这允许使用者快速、自动地生成/编辑用于驱动测试的脚本。XML API 是公开的，并且允许开发人员定义三种自己的 XML 标签句柄。

7.2　JFCUnit 基本测试方法

JFCUnit 的基本测试思路是：它提供了很多方法，可以用来模拟许多本应由传统测试人员手工进行的触发事件，如单击按钮、在文本框中输入字符或数字、鼠标双击事件等，从而实现了测试的自动化。这一优点在需要用户输入大量信息的界面测试中显得尤为重要。实际上，JFCUnit 的测试用例十分像 Robot 中的脚本的编写，使用熟练后对于自动化大批量测试十分有意义。

JFCUnit 提供两种方式来模拟用户交互：①使用 Junit.Extensions.Jfcunit.JFCTestHelper 的 EventQueue 来激活事件队列；②使用 Junit.Extensions.Jfcunit.RobotTestHelper 来封装 java.awt.Robot。无论是 JFCTestHelper 还是 RobotTestHelper，其 API 都是一样的。JFCUnit 继承了 JUnit 的测试框架，也就继承了 JUnit 的特征、性能及基本实现技术。除此之外，JFCUnit 还具有自己的独特方式和实现技术。

JFCUnit 的测试用例类似于任何其他的 JUnit 测试用例。其主要不同在于它的测试用例类应继承自基类 Junit.Extensions.Jfcunit.JFCTestCase，而不是基类 Junit.Framework.TestCase。实际上， JFCTestCase 是 TestCase 的一个子类，因此，它提供所有 Junit.Framework.TestCase 类中所期望的标准功能。显然，对于 GUI 图形界面的测试，如果仅使用此方式，将大大增加工作量，对于软件产品的开发也是不实用的。这实际上是 JFCUnit 的局限性，它只能针对扩展了 Component 的 GUI 组件进行测试。对于没有扩展的界面软件(如 ilog 这类直接由 Object 继承来的 GUI 软件)，则需要使用者自己来扩展。尽管如此，JFCUnit 的简单和易用性也应该能满足大多数实际项目测试的需要了。

7.3　JFCUnit 测试环境建立

目前最著名的 Java 集成开发环境是 IBM 的 Eclipse。因此，我们期望建立一个基于 Eclipse 的测试 Java 界面程序的 JFCUnit 环境。下面是 JFCUnit 测试环境建立的步骤。

(1) 安装 Eclipse(前面章节已介绍过)。

(2) 下载 jfcunit_Eclipse_plugin_2.08，如图 7-1 所示。一般 Eclipse 中已经内置了该插件。

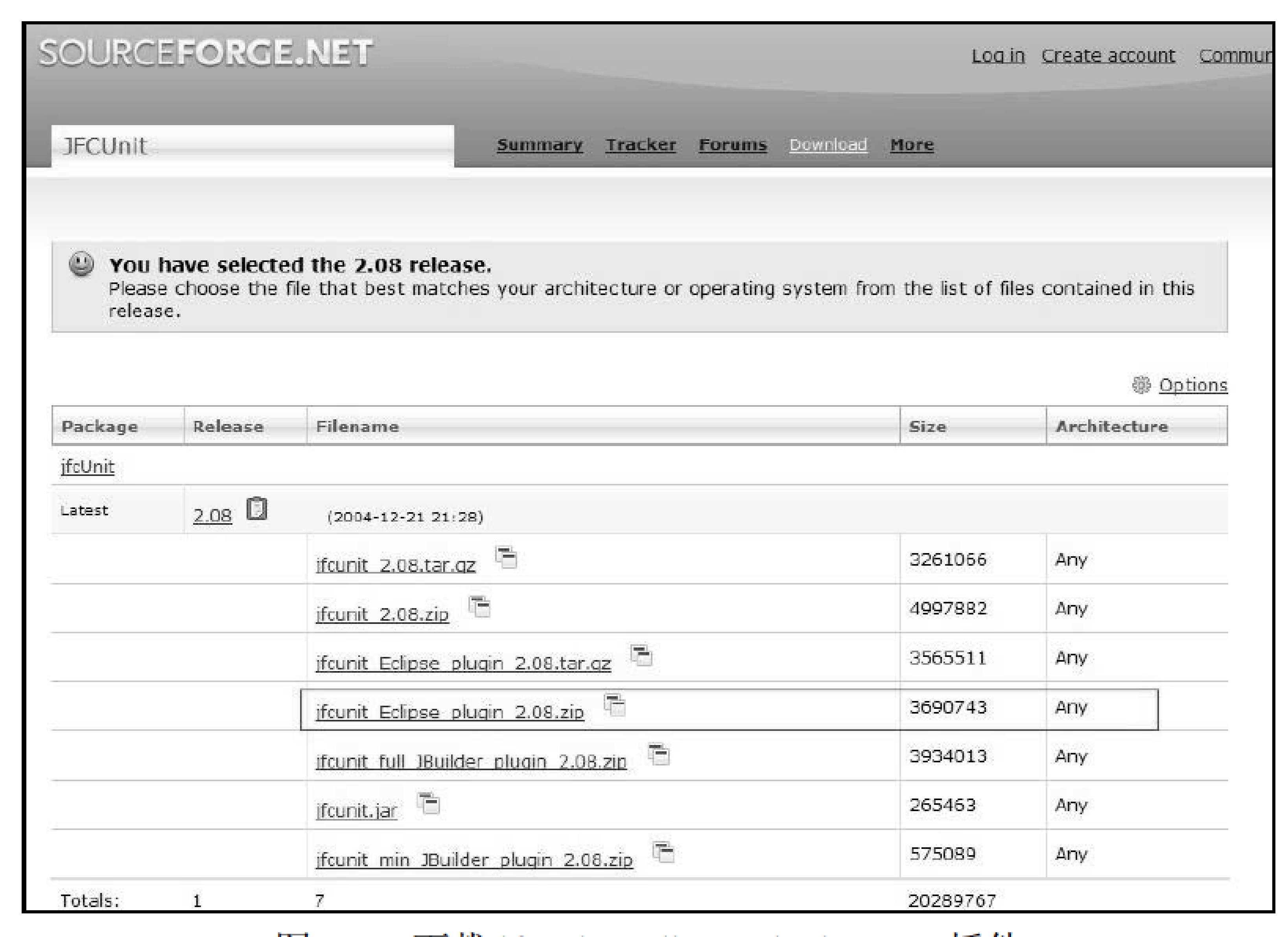


图 7-1　下载 jfcunit_Eclipse_plugin_2.08 插件

(3) 将压缩文件中的内容解压到“[路径]\eclipse\plugins”文件夹下，并保持与压缩文件中的目录结构一致，如图 7-2 所示。注意：Eclipse 会自动检测到该插件的存在。



图 7-2　安装 jfcunit_Eclipse_plugin_2.08 插件

(4) 重新启动 Eclipse，通过选择 Help|About Eclipse SDK|Plug-in Details 来查看 JFCUnit 插件是否已经安装成功，如图 7-3 所示。

(5) 在 Eclipse 上新建测试项目(如建立 JFCUnit 项目)。

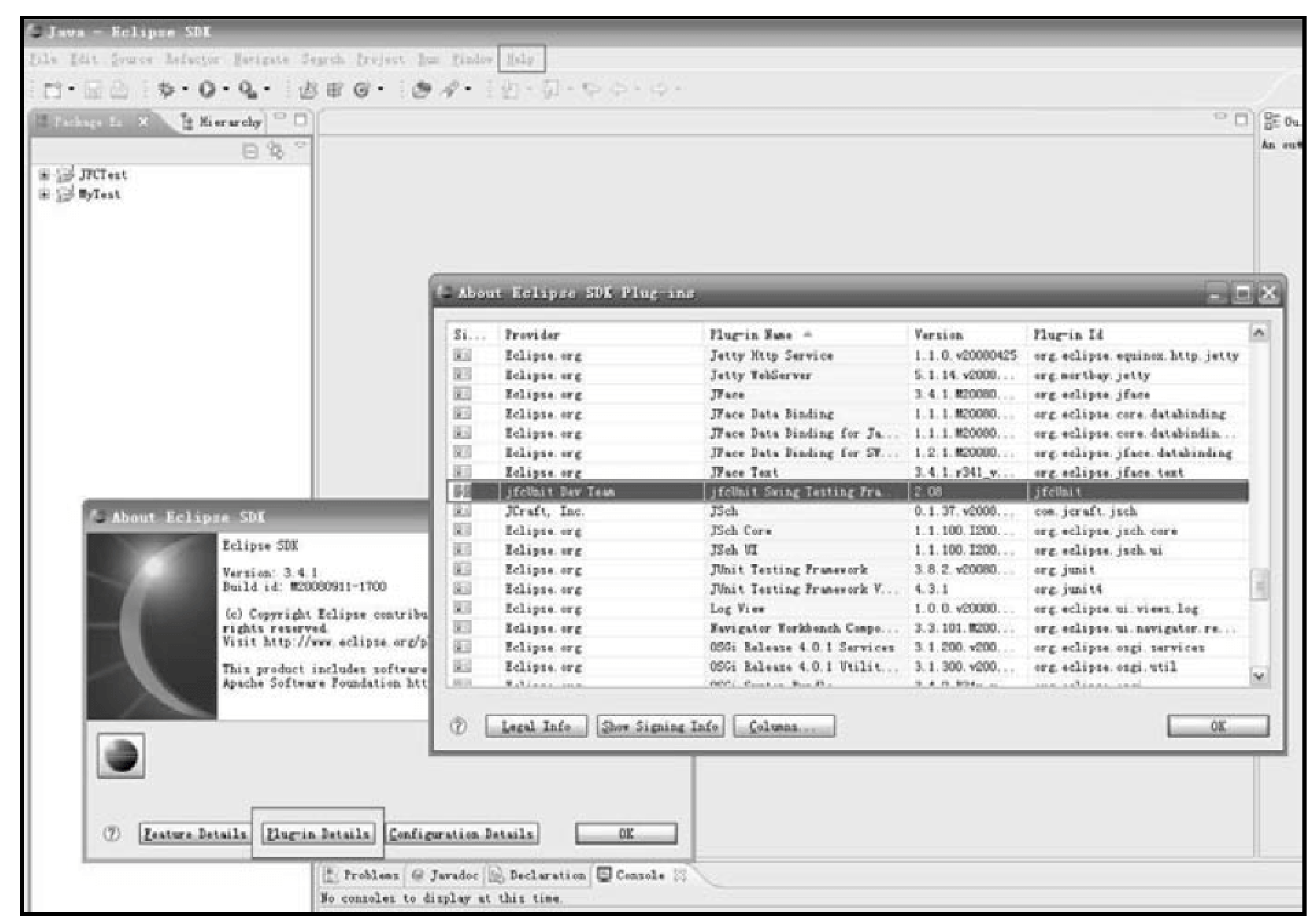


图 7-3　在 Eclipse 中检查 JFCUnit 插件

(6) 在 Eclipse 中配置路径，选择添加外部 jar，添加 jfcunit.jar 和 jakarta-regexp-1.2.jar。

注意：jakarta-regexp 是一个正则表达式 Java 包，JFCUnit 需要这个包。如果这个包没有被包含在项目中，将会产生错误，仔细检查错误提示信息，将会发现是和这个包有关。那么如何导入外部的 jar 包？

① 在项目中建立一个名为 lib 的文件夹，主要用来存放一些外部的 jar 包。

② 把相应的 jar 文件复制到这个文件夹中。

③ 在 Eclipse 下用鼠标右击项目根目录，在弹出的菜单中选择 Properties 命令。

④ 在打开的“××××的属性”(这里××××代表项目名称)对话框中，在左边的列表中选择 Java Build Path 项，然后在左边的 Tab Page 中选择 Libraries 页。

⑤ 单击 Add External JARs…，在打开的“选择 JAR”对话框中选择合适的路径和包就可以了，如图 7-4 所示。

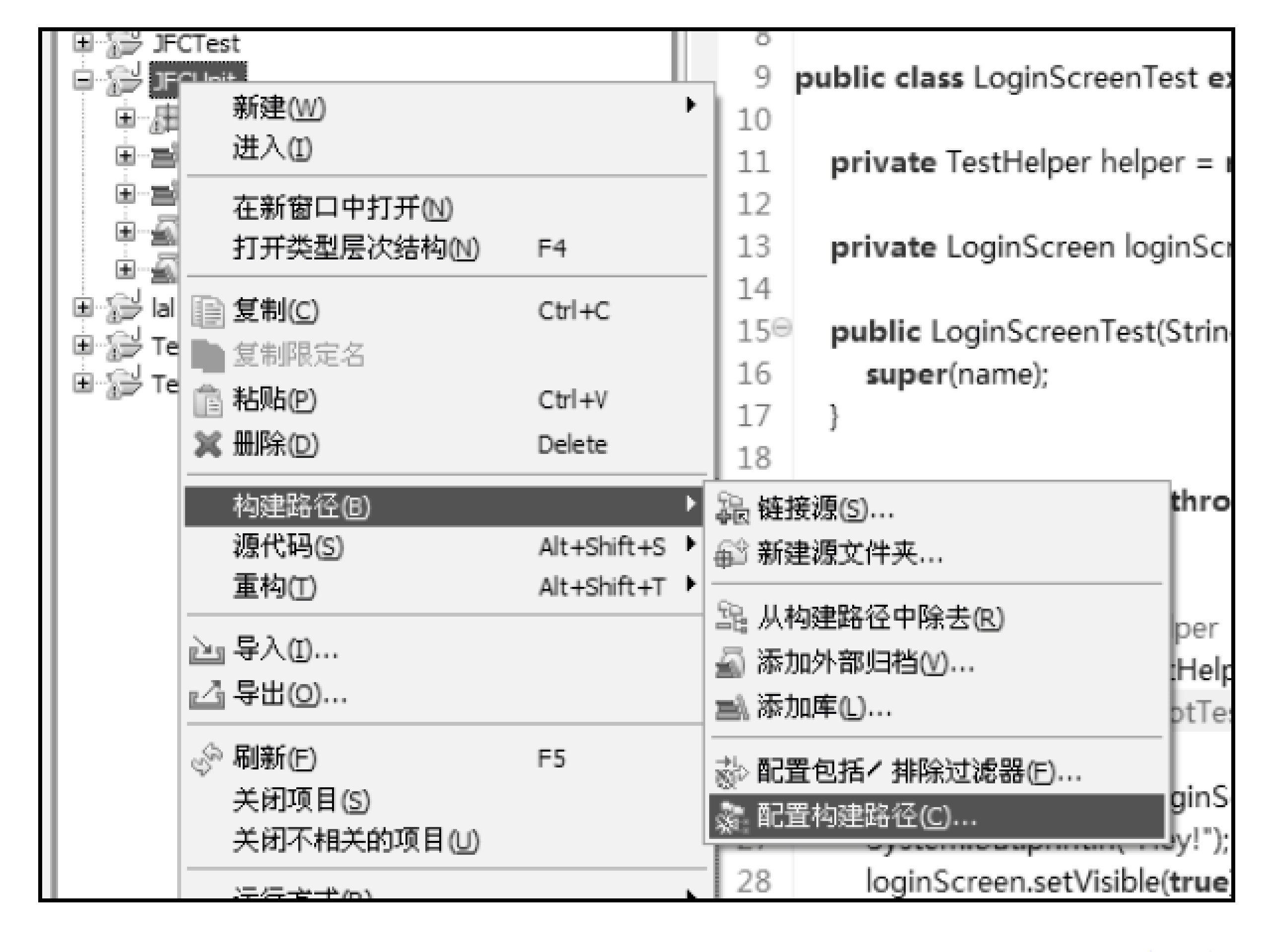




图 7-4　JFCUnit 有关 jar 包的配置

(7) 如果项目中没有包含 JUnit，就要进行导入，如图 7-5 所示。因为 JFCUnit 是继承于这个类的，没有它就会报错。



图 7-5　导入 JUnit

至此，Eclipse 下对 Java 界面程序进行测试的 JFCUnit 测试环境就被建立起来了。

7.4　JFCUnit 测试资源应用

尽管 JFCUnit 是 JUnit 测试框架的扩展，但由于 JFCUnit 是针对 Java Swing 的 GUI 测试工具，因此它具有很多属于它自身的测试资源。

7.4.1　JFCUnit 核心函数的应用方式

1. setUp()和 tearDown()

(1) 这两个函数在 JUnit 框架中用于测试的初始化和结束测试、释放资源。

(2) setUp()在每个测试方法调用前被调用，负责初始化测试方法所需要的测试环境。

(3) tearDown()在每个测试方法被调用之后被调用，负责撤销测试环境。

它们与测试方法的关系可以描述为：

测试开始→setUp()→test××××→tearDown()→测试结束

```
super.setUp( );
{
        setHelper( new JFCTestHelper( ) );
loginScreen = new LoginScreen("LoginScreenTest: " + getName( ) ); loginScreen.setVisible( true );
}
protected void tearDown( ) throws Exception
{
          loginScreen = null;
```

```
            getHelper.cleanUp( this );
            super.tearDown( );
    }
```

2. Find-Component

JFCUnit 中对对象进行测试的基础是获得测试对象实例，关键是 find…Component 系列 API 的应用。具体查找主要基于两个方面：对象组件的 Name 属性和容器中的 Index。而这两种方式都不是很直观，具体使用哪种方式在很大程度上取决于开发人员的编码，具体使用时有一定的难度和局限性。

```
public Component find(final Container cont, final int index) {
        return find(new Container[] {cont}, index);
}
```

3. assert 函数

assert 函数有以下几个。

assertNull(String message, Object object)

assertNotNull(String message, Object object)

assertEquals(String message, Object expected, Object actual)

assertTrue(String message, boolean condition)

assertFalse(String message, boolean condition)

assertSame(String message, Object expected, Object actual)

assertNotSame(String message, Object expected, Object actual)

下面是函数 assertNotNull 和 assertEquals 的实现代码。

```
static public void assertNotNull(String message, Object object) {
    assertTrue(message, object != null);
    }
static public void assertEquals(String message, Object expected, Object actual) {
        if (expected == null && actual == null)
            return;
        if (expected != null && expected.equals(actual))
            return;
        throw new ComparisonFailure(message, expected, actual);
    }
```

4. JFCTestHelper 和 TestHelper

JFCTestHelper 继承了 TestHelper 中很多用于自动化界面操作的方法，如图 7-6 所示。

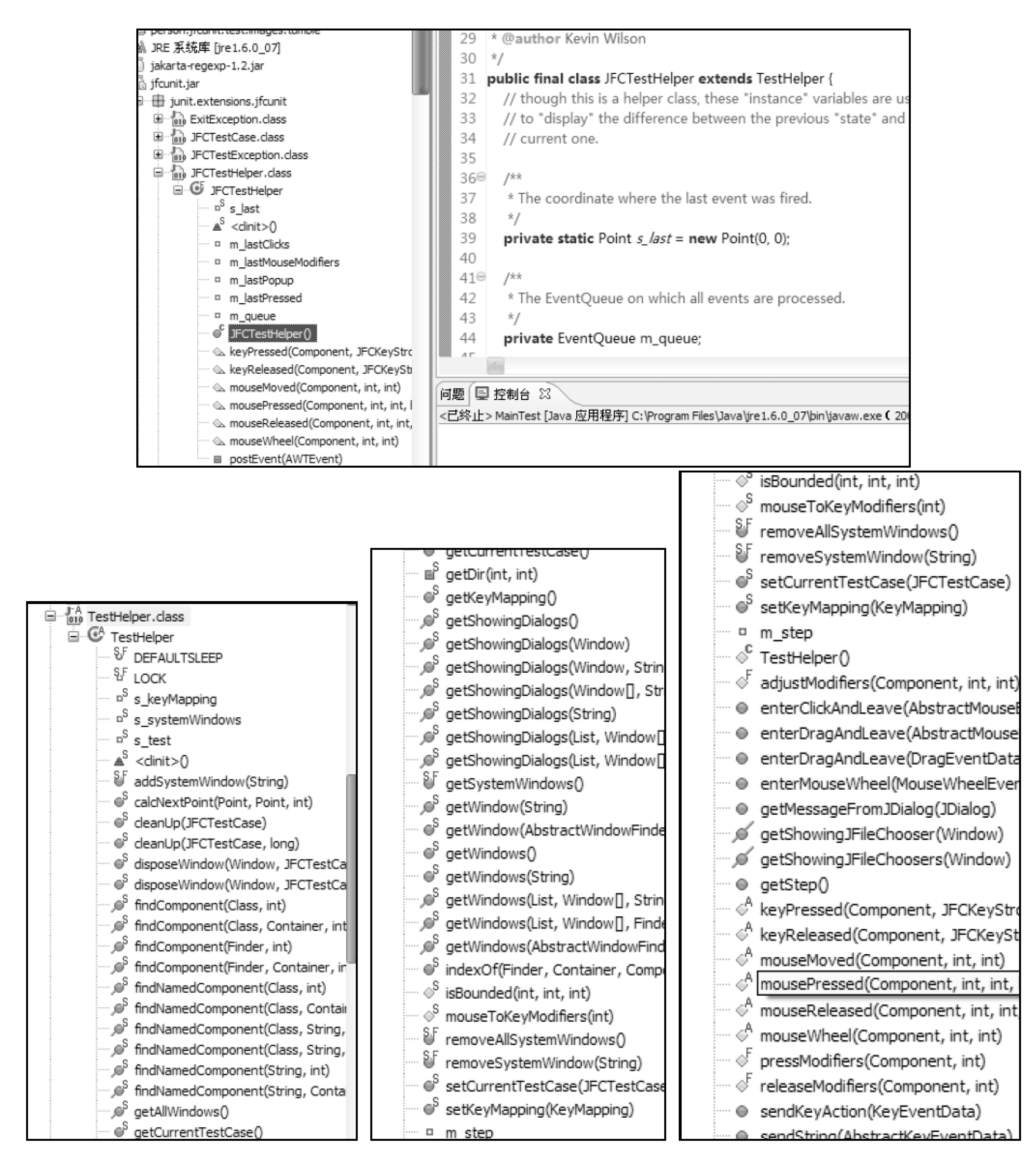


图 7-6　JFCTestHelper 从 TestHelper 继承的界面操作方法

7.4.2　JFCUnit 的界面操作要点

1. 获取界面框架的句柄

(1) 引入含有 FrameFinder 类的包：

```
import junit.extensions.jfcunit.finder.FrameFinder;
```

(2) 实例化 FrameFinder 类，通过构造函数的 String title 参数来检测界面框架的句柄(注：需要设定并了解被测代码中的框架的标题)：

```
FrameFinder frameFinder = new FrameFinder(String title);
```

(3) 调用 find()方法，获取界面框架的句柄：

```
JFrame frame = (JFrame)frameFinder.find();
```

2. 获取界面控件的句柄

(1) 引入含有 NamedComponentFinder 类的包：

```
import junit.extensions.jfcunit.finder.NamedComponentFinder;
```

(2) 实例化 NamedComponentFinder 类，通过构造函数来检测界面控件的句柄(注：需要设定并了解被测代码中的控件名，如果不了解控件类，可以 JComponent.class 作为参数)：

```
NamedComponentFinder componentFinder =
                        new NamedComponentFinder(Class cls, String name);
```

(3) 调用 find()方法，获取具体的界面控件的句柄：

```
JButton Button = (JButton)componentFinder.find();
```

3. 获取弹出式对话框的句柄

(1) 引入含有 DialogFinder 类的包：

```
import junit.extensions.jfcunit.finder.DialogFinder;
```

(2) 实例化 DialogFinder 类，通过构造函数的 String title 参数来检测对话框的句柄(注：需要设定并了解被测代码中的对话框的标题)：

```
DialogFinder dFinder = new DialogFinder(String title);
```

(3) 调用 find()方法，获取对话框的句柄：

```
JDialog dialog = (JDialog)dFinder.find();
```

4. 设置触发器和基本触发事件

(1) 引入含有 TestHelper、JFCTestHelper 类的包：

```
import junit.extensions.jfcunit.TestHelper;
import junit.extensions.jfcunit.JFCTestHelper;
```

(2) 声明 TestHelper 类的对象，并通过 JFCTestHelper 类进行实例化：

```
TestHelper helper = new JFCTestHelper();
```

(3) 调用 setHelper(TestHelper helper)方法，装载事件触发器：

```
setHelper(helper);
```

(4) 调用相关方法，模拟相应的基本触发事件。

① 鼠标单击事件：

```
helper.enterClickAndLeave(AbstractMouseEventData arg0);
```

② 鼠标滚轮事件：

```
helper.enterMouseWheel(MouseWheelEventData arg0);
```

③ 多选事件：

```
helper.enterDragAndLeave(DragEventData arg0);
```

④ 键盘按键事件：

```
helper.sendKeyAction(KeyEventData evtData);
```

⑤ 键盘输入事件：

```
helper.sendString(AbstractKeyEventData arg0);
```

5. 设置事件触发源和具体触发动作

(1) 引入含有各种事件源类的包：

```
import junit.extensions.jfcunit.eventdata.*;
```

(2) 实例化相应的事件源类，获得事件触发源，通过构造函数的参数来设定具体触发动作(注：用蓝色标识的是可选参数)。

① 鼠标单击事件：

```
new MouseEventData(JFCTestCase testCase, Component comp,
                   int numberOfClicks, boolean isPopupTrigger);
```

numberOfClicks：鼠标单击次数，默认为一次。

isPopupTrigger：true 为单击右键；false 为单击左键，默认为 false。

② 键盘按键事件：

```
new KeyEventData(JFCTestCase testCase, Component comp, int keyCode);
```

③ 键盘输入事件：

```
new StringEventData(JFCTestCase testCase, Component comp, String string);
```

④ 多选事件：

```
new DragEventData(JFCTestCase testCase, AbstractMouseEventData source,
```

```
                    AbstractMouseEventData dest);
```

⑤ 下拉列表选中项事件：

```
new JComboBoxMouseEventData(JFCTestCase testCase, JComboBox comboBox,
                            Object element, int numberOfClicks);
```

⑥ 列表中选中和单击行的事件：

```
new JListMouseEventData(JFCTestCase testCase, JList list, int elementIndex, int numberOfClicks);
```

⑦ 切换和单击 Tab 标签事件：

```
new JTabbedPaneMouseEventData(JFCTestCase testCase,JTabbedPane tabPane,
                              int tabIndex, int numberOfClicks);
```

⑧ 表格内的单元格选中和单击事件：

```
new JTableMouseEventData(JFCTestCase testCase, JTable table,
                         int rowIndex, int columnIndex,
                         int numberOfClicks);
```

⑨ 选中和单击树节点事件：

```
new JTreeMouseEventData(JFCTestCase testCase, JTree tree, String nodeValue, int numberOfClicks);
```

(3) 将事件触发源和具体触发动作载入事件触发器相应的基本触发事件中。

7.4.3　JFCUnit 中主要的 GUI 类

1. GUI 组件发现类

1) 通用命名组件发现类

```
NamedComponentFinder(java.lang.Class cls，java.lang.String name)
```

这个类使用得最多，也最容易，但需要在 GUI 程序中将组件命名(如 cancelBtn.setName ("cancelBtn")。

示例代码：

```
NamedComponentFindercancelBtnFinder =
              new NamedComponentFinder(JButton.class，  "cancelBtn");
JButton cancelBtn = (JButton) cancelBtnFinder.find();
assertNotNull("cancelBtn not found!"，  cancelBtn);
assertEquals(true，  cancelBtn.isEnabled());
```

2) 其他组件发现类

这些类主要是针对一些特定的 GUI 组件提供了一些特殊的查找方法。主要有DialogFinder、FrameFinder、JLabelFinder等。具体使用可以查找 API 文档，在 junit.extensions.jfcunit 下的 doc 文档中。

示例代码：

```
FrameFinder frameFinder = new FrameFinder("FrameDemo"); // 字符串为 Frame 标题
List frames = frameFinder.findAll();
assertEquals("frames size wrong"，1，frames.size());
JFrame frame = (JFrame)frames.get(0);
assertNotNull("frame is null !"，frame);
```

2. 用户动作模拟类

这些类主要有 AbstractMouseEventData、DragEventData、JComboBoxMouseEventData、JListMouseEventData、JTabbedPaneMouseEventData 等。它们在使用上大同小异。下面以表格为例：

```
finder.setName( "table" );
JTable table = (JTable)finder.find(frame，0); // 从 Frame 中查找到表格
assertNotNull( "table not found !" ，table);
// 模拟双击单元格
JTableMouseEventDatatableEvent= new JTableMouseEventData( this，table，1，2，2);
helper .enterClickAndLeave(tableEvent);
this .flushAWT(); // 执行事件派发队列中的线程，保证事件已经被响应。
// 模拟改变单元格的文本值
helper .enterClickAndLeave(tableEvent);
this .flushAWT();
helper.sendKeyAction(new KeyEventData(this，tableEvent.getComponent()，KeyEvent.VK_DELETE));
helper .sendString( new StringEventData( this，tableEvent.getComponent()，"" ));
helper .sendString( new StringEventData( this，tableEvent.getComponent()，"wukaichun" ));
this .flushAWT();
// 模拟多选表格行
JTableMouseEventDatasrcEvent = new JTableMouseEventData( this，table，1，2，1);
JTableMouseEventDatasinkEvent = new JTableMouseEventData( this，table，2，2，1);
helper .enterDragAndLeave( new DragEventData( this，srcEvent，sinkEvent));
this .flushAWT();
```

7.5　JFCUnit 测试应用举例

JFCUnit 测试流程如图 7-7 所示。

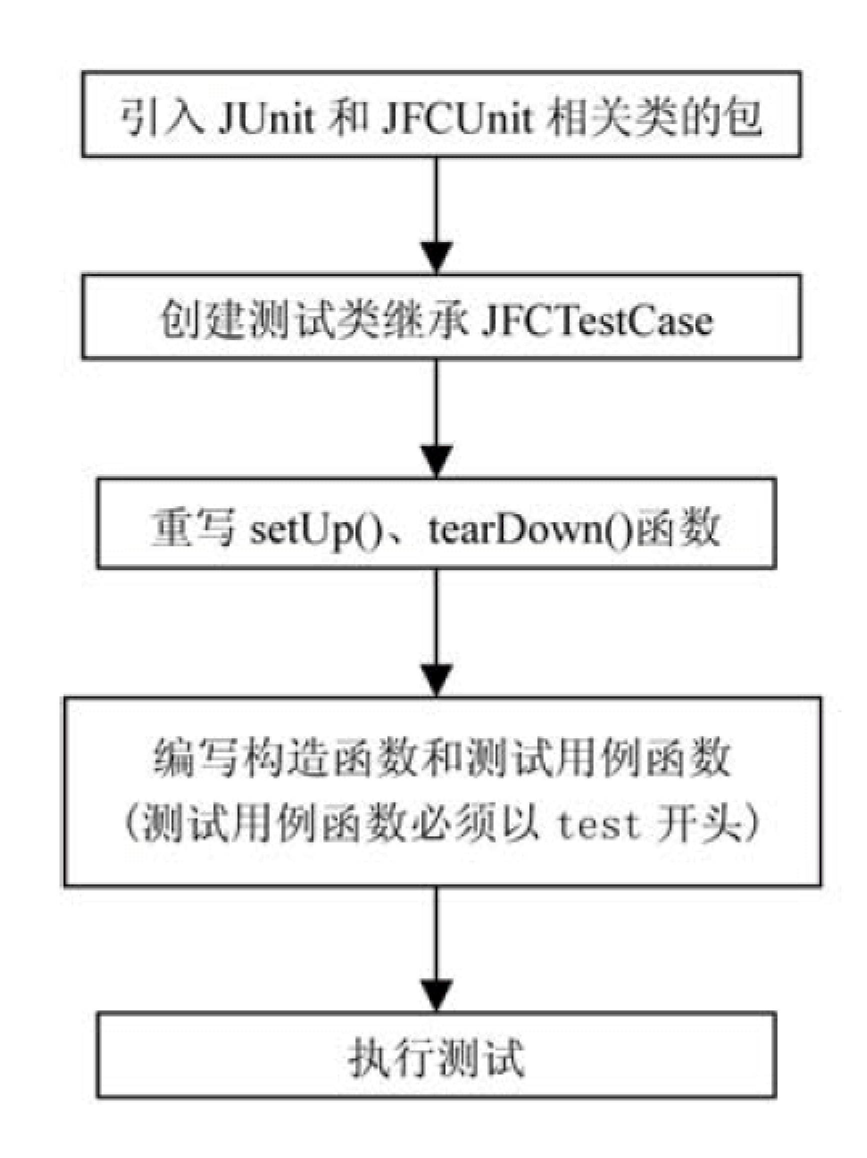


图 7-7　JFCUnit 的测试流程

对于实际的 GUI 项目，往往需要按界面分别进行测试。为了使多个测试用例能共用测试环境(不必每次都启动项目)，可以将被测工程与 JUnit 一起启动，而后在 JUnit 的 SwingUI 中选取用例执行。

例如：

```
package person.jfcunit.test;
import junit.swingui.TestRunner;
public class MainTest {
public static void main(String[] args) {
      TreeIconDemo.main( new String[]{}); // 启动被测 GUI 程序
      TestRunner.main( new String[]{}); // 启动 JUnit
   }
}
```

注意：启动配置中除主函数变为 MainTest 外，其他配置与被测工程都一样；如果使用 JUnit SwingUI 测试界面，为了保证所有的组件都为同一加载器加载，需要将被测类改为默认加载器加载。

(1) 在 junit.jar 的 junit.runner 下，找到文件 excluded.properties。

(2) 修改 excluded.properties，将被测类添加到 excluded 中，如下所示：

```
#
```

```
# The list of excluded package paths for the TestCaseClassLoader
#
excluded.0=sun.*
excluded.1=com.sun.*
excluded.2=org.omg.*
excluded.3=javax.*
excluded.4=sunw.*
excluded.5=java.*
excluded.6=org.w3c.dom.*
excluded.7=org.xml.sax.*
excluded.8=net.jini.*
excluded.9=org.apache.commons.logging.*
excluded.9=person.*
```

这里列出的包将不使用 TestCaseClassLoader 加载，而是由默认加载器加载。如最后一行“excluded.9=person.* ”，将 person 包中的类排除在 TestCaseClassLoader 加载之外。这个时候，运行 MainTest，就可以对被测项目进行测试了。

另一种启动界面功能测试的方法是，在测试类的构造函数中调用被测界面代码的 main()函数。下面以 TreeIconDemo(如图 7-8 所示)的界面功能测试为例进行介绍。

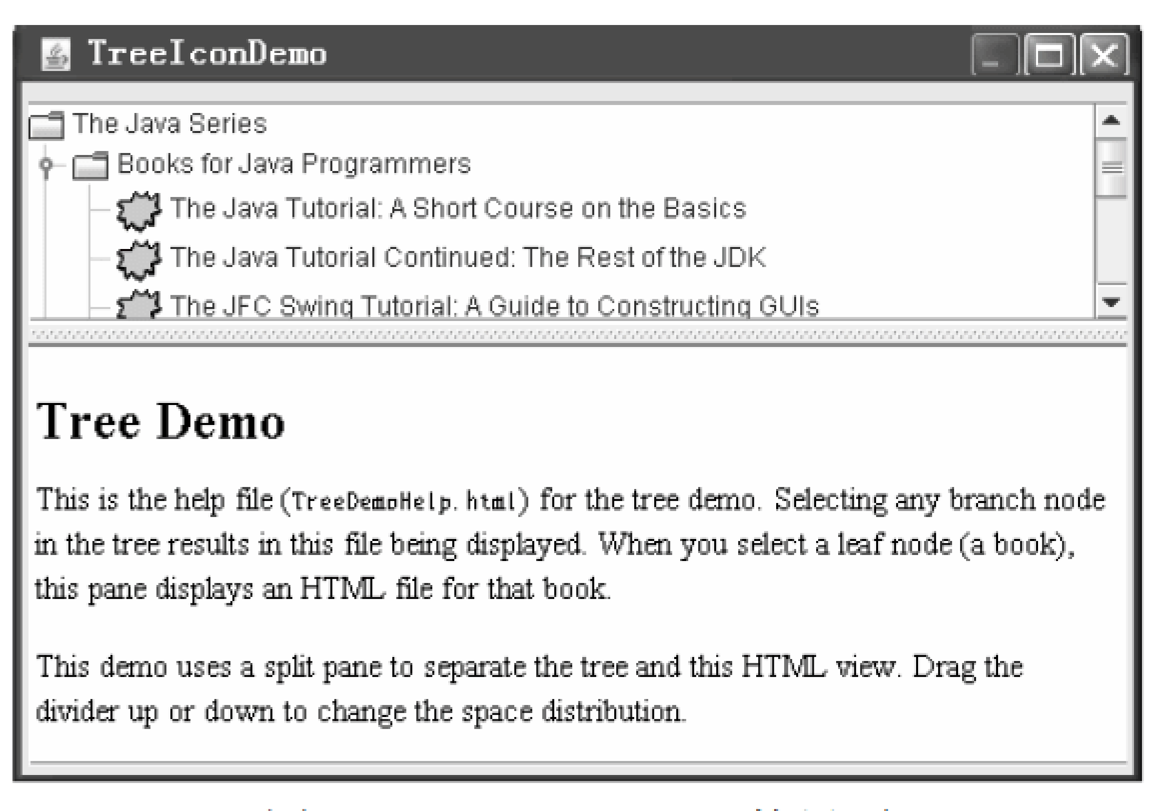


图 7-8　TreeIconDemo 的界面

```
/**
 * TreeIconDemo application that requires the following additional files:
 *      TreeDemoHelp.html
 *      arnold.html
 *      bloch.html
 *      chan.html
 *      jls.html
 *      swingtutorial.html
 *      tutorial.html
 *      tutorialcont.html
 *      vm.html
```

```
 */
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.ImageIcon;
import java.net.URL;
import java.io.IOException;
import java.awt.Dimension;
import java.awt.GridLayout;
public class TreeIconDemo extends JPanel
                                        implements TreeSelectionListener {
    private JEditorPane htmlPane;
    private JTree tree;
    private URL helpURL;
    private static boolean DEBUG = false;
    public TreeIconDemo() {
        super(new GridLayout(1,0));
        //Create the nodes.
        DefaultMutableTreeNode top =
            new DefaultMutableTreeNode("The Java Series");
        createNodes(top);
        //Create a tree that allows one selection at a time.
        tree = new JTree(top);
        tree.setName("tree");
        tree.getSelectionModel().setSelectionMode
                (TreeSelectionModel.SINGLE_TREE_SELECTION);
        //Set the icon for leaf nodes.
        ImageIcon leafIcon = createImageIcon("images/middle.gif");
        if (leafIcon != null) {
            DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
            renderer.setLeafIcon(leafIcon);
            tree.setCellRenderer(renderer);
        } else {
            System.err.println("Leaf icon missing; using default.");
```

```
        }
        //Listen for when the selection changes.
        tree.addTreeSelectionListener(this);
        //Create the scroll pane and add the tree to it.
        JScrollPane treeView = new JScrollPane(tree);
        //Create the HTML viewing pane.
        htmlPane = new JEditorPane();
        htmlPane.setEditable(false);
        initHelp();
        JScrollPane htmlView = new JScrollPane(htmlPane);
        //Add the scroll panes to a split pane.
        JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        splitPane.setTopComponent(treeView);
        splitPane.setBottomComponent(htmlView);
        Dimension minimumSize = new Dimension(100, 50);
        htmlView.setMinimumSize(minimumSize);
        treeView.setMinimumSize(minimumSize);
        splitPane.setDividerLocation(100); //XXX: ignored in some releases
                                           //of Swing. bug 4101306
        //workaround for bug 4101306:
        //treeView.setPreferredSize(new Dimension(100, 100));
        splitPane.setPreferredSize(new Dimension(500, 300));
        //Add the split pane to this panel.
        add(splitPane);
    }
    /** Required by TreeSelectionListener interface. */
    public void valueChanged(TreeSelectionEvent e) {
        DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                              tree.getLastSelectedPathComponent();
        if (node == null) return;
        Object nodeInfo = node.getUserObject();
        if (node.isLeaf()) {
            BookInfo book = (BookInfo)nodeInfo;
            displayURL(book.bookURL);
            if (DEBUG) {
                System.out.print(book.bookURL + ":   \n       ");
            }
        } else {
            displayURL(helpURL);
        }
        if (DEBUG) {
            System.out.println(nodeInfo.toString());
```

```
        }
    }
    private class BookInfo {
        public String bookName;
        public URL bookURL;
        public BookInfo(String book, String filename) {
            bookName = book;
            bookURL = TreeIconDemo.class.getResource(filename);
            if (bookURL = = null) {
                System.err.println("Couldn't find file: "
                                   + filename);
            }
        }
        public String toString() {
            return bookName;
        }
    }
    private void initHelp() {
        String s = "TreeDemoHelp.html";
        helpURL = TreeIconDemo.class.getResource(s);
        if (helpURL = = null) {
            System.err.println("Couldn't open help file: " + s);
        } else if (DEBUG) {
            System.out.println("Help URL is " + helpURL);
        }
        displayURL(helpURL);
    }
    private void displayURL(URL url) {
        try {
            if (url != null) {
                htmlPane.setPage(url);
            } else { //null url
     htmlPane.setText("File Not Found");
                if (DEBUG) {
                    System.out.println("Attempted to display a null URL.");
                }
            }
        } catch (IOException e) {
            System.err.println("Attempted to read a bad URL: " + url);
        }
    }
    private void createNodes(DefaultMutableTreeNode top) {
```

```
DefaultMutableTreeNode category = null;
DefaultMutableTreeNode book = null;
category = new DefaultMutableTreeNode("Books for Java Programmers");
top.add(category);
//original Tutorial
book = new DefaultMutableTreeNode(new BookInfo
      ("The Java Tutorial: A Short Course on the Basics",
      "tutorial.html"));
category.add(book);
//Tutorial Continued
book = new DefaultMutableTreeNode(new BookInfo
      ("The Java Tutorial Continued: The Rest of the JDK",
      "tutorialcont.html"));
category.add(book);
//JFC Swing Tutorial
book = new DefaultMutableTreeNode(new BookInfo
      ("The JFC Swing Tutorial: A Guide to Constructing GUIs",
      "swingtutorial.html"));
category.add(book);
//Bloch
book = new DefaultMutableTreeNode(new BookInfo
       ("Effective Java Programming Language Guide",
        "bloch.html"));
category.add(book);
//Arnold/Gosling
book = new DefaultMutableTreeNode(new BookInfo
      ("The Java Programming Language", "arnold.html"));
category.add(book);
//Chan
book = new DefaultMutableTreeNode(new BookInfo
      ("The Java Developers Almanac",
      "chan.html"));
category.add(book);
category = new DefaultMutableTreeNode("Books for Java Implementers");
top.add(category);
//VM
book = new DefaultMutableTreeNode(new BookInfo
       ("The Java Virtual Machine Specification",
       "vm.html"));
category.add(book);
//Language Spec
book = new DefaultMutableTreeNode(new BookInfo
```

```
                ("The Java Language Specification",
                "jls.html"));
            category.add(book);
        }
        /** Returns an ImageIcon, or null if the path was invalid. */
        protected static ImageIcon createImageIcon(String path) {
            java.net.URL imgURL = TreeIconDemo.class.getResource(path);
            if (imgURL != null) {
                return new ImageIcon(imgURL);
            } else {
                System.err.println("Couldn't find file: " + path);
                return null;
            }
        }
        /**
         * Create the GUI and show it.  For thread safety,
         * this method should be invoked from the
         * event-dispatching thread.
         */
        private static void createAndShowGUI() {
            //Create and set up the window.
            JFrame frame = new JFrame("TreeIconDemo");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            //Create and set up the content pane.
            TreeIconDemo newContentPane = new TreeIconDemo();
            newContentPane.setOpaque(true); //content panes must be opaque
            frame.setContentPane(newContentPane);
            //Display the window.
            frame.pack();
            frame.setVisible(true);
        }
        public static void main(String[] args) {
            //Schedule a job for the event-dispatching thread:
            //creating and showing this application's GUI.
            javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    createAndShowGUI();
                }
            });
        }
    }
```

下面是测试用例 TreeIconDemoTest.java 的代码：

```
import java.awt.Component;
import java.util.List;
import javax.swing.JFrame;
import javax.swing.JTree;
import junit.extensions.jfcunit.JFCTestCase;
import junit.extensions.jfcunit.JFCTestHelper;
import junit.extensions.jfcunit.TestHelper;
import junit.extensions.jfcunit.eventdata.JTreeMouseEventData;
import junit.extensions.jfcunit.finder.FrameFinder;
import junit.extensions.jfcunit.finder.NamedComponentFinder;
public class TreeIconDemoTest extends JFCTestCase {
    TestHelper helper = null;
    public TreeIconDemoTest(String name) {
        super(name);
        helper = new JFCTestHelper();
        this.setHelper(helper);
        TreeIconDemo.main(new String[]{});
    }
    public void testUI()
    {
        FrameFinder frameFinder = new FrameFinder("TreeIconDemo");
        List frames = frameFinder.findAll();
        assertEquals("frames size wrong",1,frames.size());
        JFrame frame = (JFrame)frames.get(0);
        assertNotNull("frame is null !",frame);
        NamedComponentFinder finder = new NamedComponentFinder(Component.class,"");
        finder.setName("tree");
        JTree tree = (JTree)finder.find(frame,0);
        //双击 Books for Java Implementers 节点
        helper.enterClickAndLeave(new JTreeMouseEventData(this,tree,"Books for Java Implementer",2));
        this.flushAWT();
        //继续单击 The Java Language Specification 节点
        helper.enterClickAndLeave(new JTreeMouseEventData(this,tree,"The Java Language Specification",1));
        this.flushAWT();
        this.pause();
    }
    protected void setUp() throws Exception {
        super.setUp();
    }
    protected void tearDown() throws Exception {
        super.tearDown();
    }
}
```

部分代码分析：

```
helper.enterClickAndLeave(new JTreeMouseEventData(this,tree,"Books for Java Implementer",2));
this.flushAWT();
 //双击 Books for Java Implementers 节点
     helper.enterClickAndLeave(new JTreeMouseEventData(this,tree,"The Java Language Specification",1));
this.flushAWT();
//继续单击 The Java Language Specification 节点
List frames = frameFinder.findAll();
//发现所有的组件
assertEquals("frames size wrong",1,frames.size());
//测试组件的大小，如果不符合要求，则显示"frames size wrong"
assertNotNull("frame is null !",frame);
//判断组件是否为空，若为空，则显示"frame is null !"
```

执行上面的被测代码 TreeIconDemo.java，用测试代码 TreeIconDemoTest.java 进行测试，方法如图 7-9 所示(选择 JUnit Test 命令进行测试)。

(1) 生成的进度条为绿色，说明测试成功。

(2) 生成的进度条为红色，说明被测代码出现错误。下面窗格中为测试人员列出了 Failure Trace。

(3) 若生成失败，则进度条为蓝色。

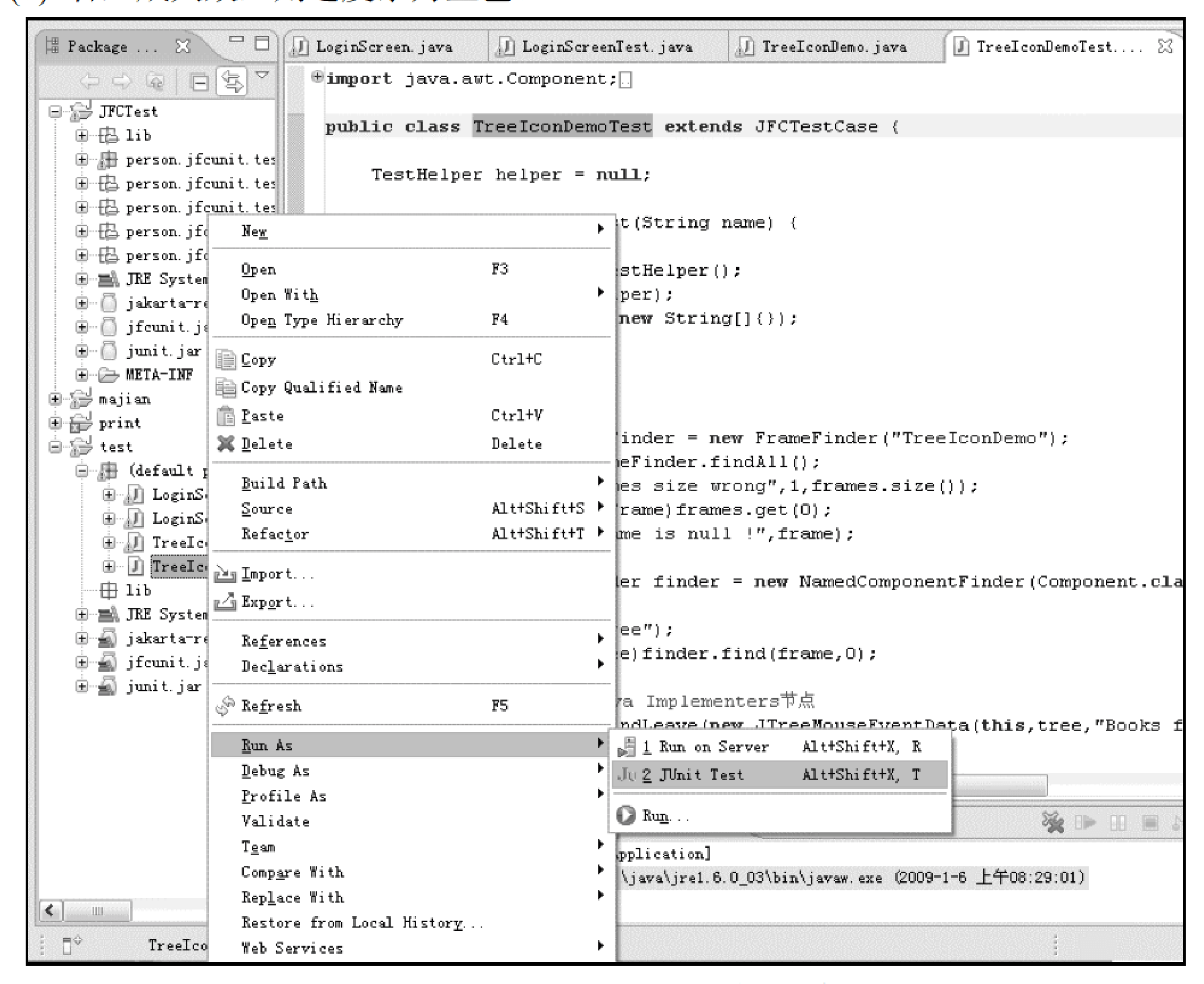
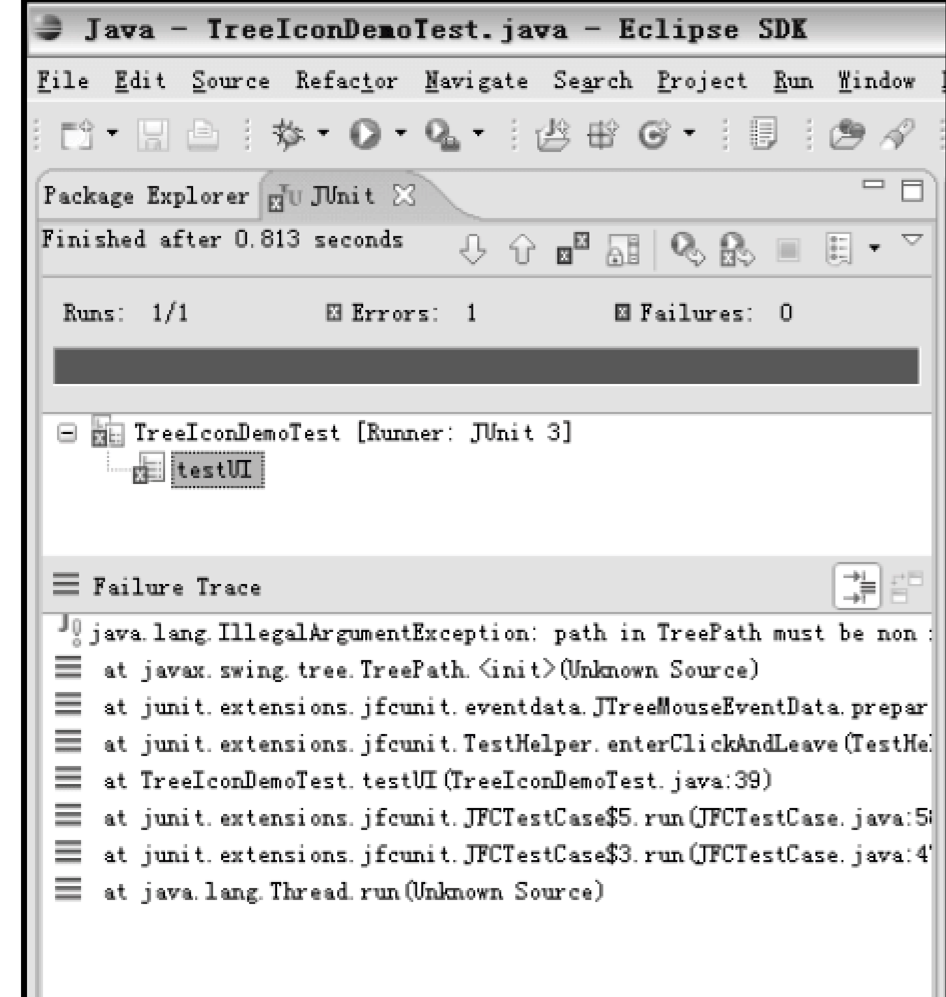


图 7-9　TreeIconDemo 测试结果分类



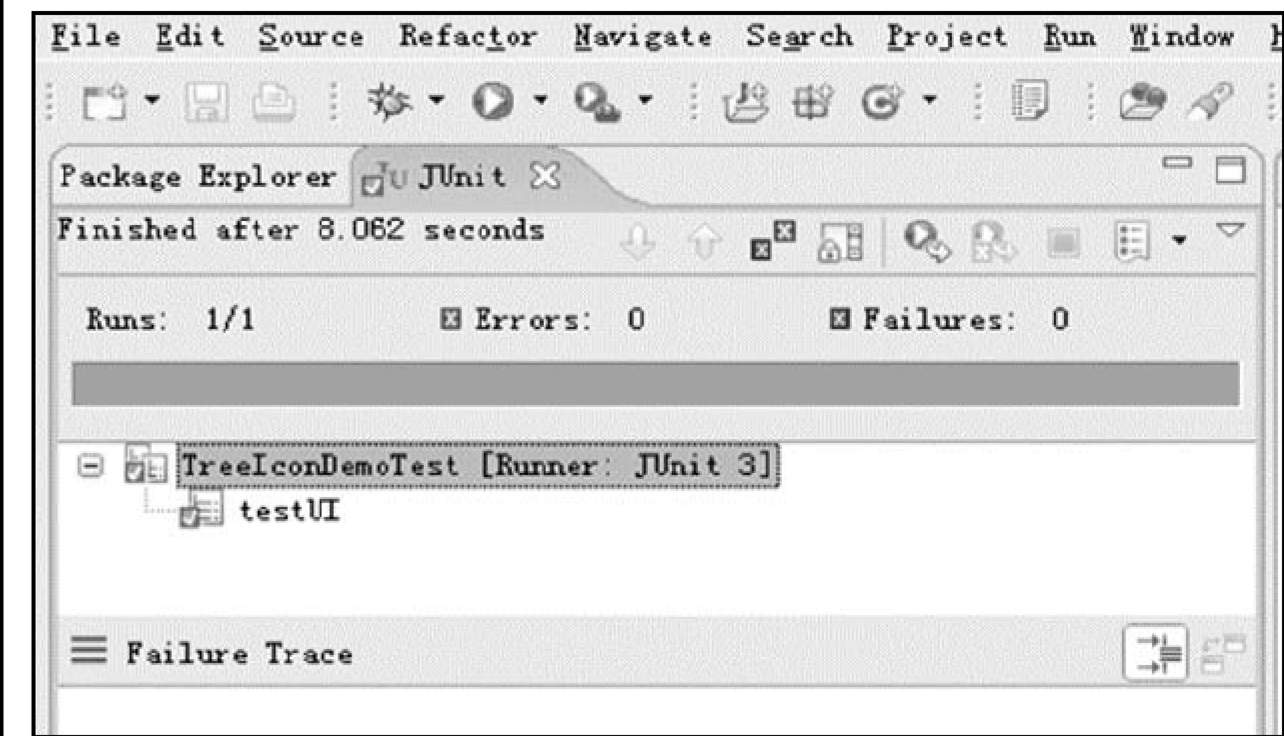


图 7-9　(续)

7.6　JFCUnit XML 测试框架

JFCUnit XML 是一个建立在 XML 框架之上的测试框架，如图 7-10 所示。XML 框架同时支持 Swing 和非 Swing 的测试。JFCUnit XML 允许开发人员编写 XML 代码来开发用于各种测试目的的测试集。可通过录制成为 XML 的方式使大多数的测试生成过程自动化。这种实现并非需要删除所有的 Java 代码。因为我们并不是要全部替换 Java 代码，而是要提供定义和使用测试用例来开展 GUI 部件测试的有效手段。用 XML 进行录制可使我们重复应用这些代码，也就是说，在录制代码的基础上可以很方便地插入和修改代码。事实上，这使得用户能够快速地自动生成可重新编辑的脚本去进行测试。

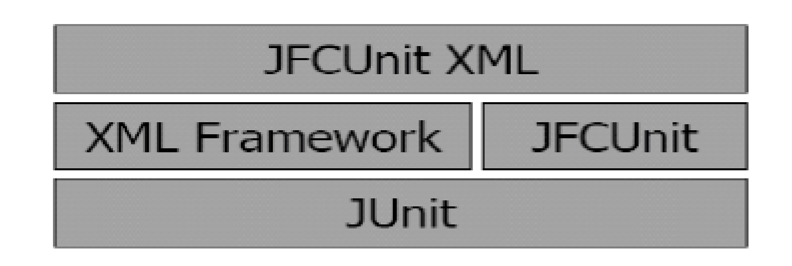


图 7-10　JFCUnit XML 测试框架

1. JFCUnit XML 基础

XML 是一种基于 SGML 的标记语言，已成为网络数据交互的一种强大的工具语言。XML 的特性非常符合 JFCUnit 的 GUI 测试特点，它提供了定义和处理 GUI 测试用例的一个有效方法，并在 JFCUnit 类中开发了 XML 的执行接口：XMLTestCase(XMLTestSuite)。

为了开发 JFCUnit 类的 XML 接口，开发了一个基于 XML JUnit 的框架以提供能够被其他项目重用的构件。该框架可通过 XML 来定义有关特性。这些特性在它们的作用域范围内允许它们在测试用例之间使用。定义在 XML 中的各个测试用例可以访问定义在父类或祖父类的特性。父级可通过指定特性名来隐式访问，也可以通过前缀特性名并加上一组

合适的“../”父访问路径来显式访问。特性能够用宏“${name}”中的具体属性值来替换“name”，而且这种替换可以组合起来以产生新的属性值。

程序可在测试集和测试用例层面上定义，这些程序遵循与特性同样的作用域规则。父访问也能够用在更高作用域级别的程序定义上，这允许生成程序库。

这里保留了两个程序名：“setup”为 JUnit 的 TestCase.setUp()方法，“teardown”为 JUnit 的 TestCase.tearDown()方法。

测试开发人员很容易就能扩展 XML 接口，特定的 Tag Handler(标签句柄)可结合 Java 代码的后端开发出来并写入到 XML 中。这些特定的 Tag Handler 可通过 XML 脚本定义进行注册/不注册。通过 JFC XML 框架定义的持久 Tag Handler 可重写或恢复。

下面介绍一下 Tag Handler 的概念。

(1) 标签集(Suite Tags)。是一组测试用例或测试集的集合。该集合包括三类要素：①引入定义在其他 XML 文件中的有关集的文件标签；②程序标签定义；③在测试用例间共享的特性标签。

(2) 用文件标签组织测试：Suite of Suites(如图 7-11 所示)。

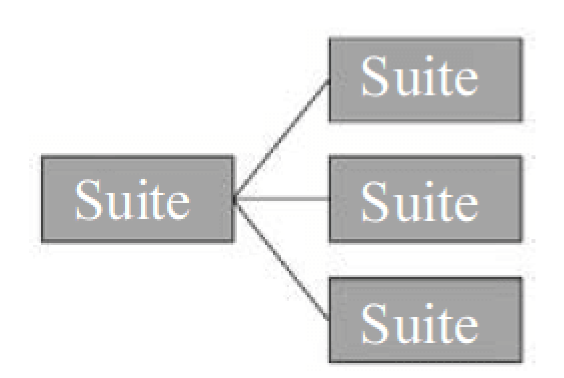


图 7-11　Suite of Suites

```
<suite name="Abc">
<file name="a.xml"/>
<file name="b.xml"/>
<file name="subdir/c.xml"/>
</suite>
```

(3) 程序标签用法(如图 7-12 所示)。

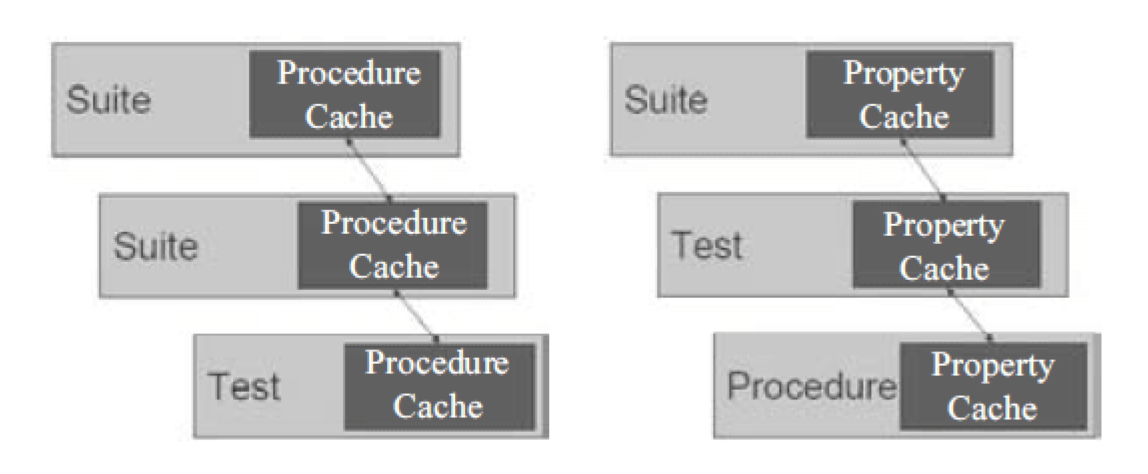


图 7-12　程序标签组织

```
<suite name="Procedure Example">
    <procedure name="Login">
    …
    </procedure>
    <procedure name="Logout">
```

```
    …
    </procedure>
    <test name="Login">
    <procedure call="Login"/>
    …
    <procedure call="Login"/>
    </test>
</suite>
```

(4) 特性标签。

特性标签包含有：

① 标签名对应于相应值。

② 作用域。

③ 用下面文法所表示的 XML 属性值对特性标签进行访问：

```
${name} syntax
```

④ 用法：

```
<property name="debug"
value="true"/>
```

(5) 其他定义。

① 向程序传递数据(in/out)。

② 判定标签进行分支跳转。

③ 对各种标签进行循环处理。

下面给出一个 XML 基本概念的例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="My Test Cases">
    <!-- Define a doc tag handler to allow the embed doc into the test case. -->
<taghandlers action="add" tagname="doc"
classname="junit.extensions.xml.elements.NoOpTagHandler"/>
    <!-- A Suite level procedure -->
    <procedure name="setUp">
        <!-- Show popup with "Starting Test" -->
        <echo message="Starting Test" mode="dialog/>
    </procedure>
    <procedure name="tearDown">
        <!-- Show popup with "Test Complete" -->
        <echo message="Test Complete" mode="dialog/>
    </procedure>
    <!-- Load a procedure library -->
```

```
    <file name="library.xml"/>
    <suite name="My Internal test suite">
        <!-- a procedure defined in a internal suite -->
        <procedure name="Another procedure">
            <!-- return a value to the calling test case -->
            <!-- to the value of the result property -->
            <property name="../suitetest" value="${result}"/>
            ...
        </procedure>

        <test name="Test Procedure">
            <procedure name="setUp">
                <!-- one ../ takes us to the test case scope-->
                <!-- two ../ takes us to the test suite scope -->
                <procedure call="../../setUp"/>
            </procedure>
            <procedure name="TEST">
                ...
            </procedure>
            <doc>
                This is a custom tag handler and should be
                ignored since the doc is mapped to the NoOp
                (No Operation) tag handler.
            </doc>
            <!-- example of calling and passing a property -->
            <procedure call="TEST" result="true"/>
            <procedure call="Another procedure"/>
        </test>
    </suite>
</suite>
```

程序库例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Library">
    <!-- Define a custom tag handler -->
    <!-- Tag handlers are global in nature. Thus they are not scoped. -->
<taghandlers action="add" tagname="mytagHandler"
classname="customPackage.CustomTagHandler"/>
    <!-- A Library procedure -->
    <!-- NOTE: the subtlety of using the ../ to place the Login procedure -->
    <!--          into the parent test suite scope. Without this the          -->
    <!--          procedure would only be visible to tests and suites          -->
```

```
    <!--        contained in this XML.                               -->
    <procedure name="../Login">
      <mytagHandler .../>
        ...
    </procedure>
    <!-- A Library procedure -->
    <procedure name="../Logout">
        ...
    </procedure>
</suite>
```

2. 生成 XML 语言元素

可以生成多个语言元素以支持 XML 测试脚本的生成。

(1) Choose/When/Otherwise：接受由 XLS 转换到允许在 XML 测试用例中存在分支逻辑的 Choose/When/Otherwise 构造。

(2) Echo：允许将正文输出到 stdout/stderr 对话框，而且允许设置断言，该对话框的模式将用于测试用例的同步执行。

(3) Noop：noop tag handler 用于其他作为测试用的 tag handlers 开关。

(4) Assert/Fail：断言确定的特性。

(5) Evaluate：计算空指针方法并存放有关值到一个特性中。

(6) Save：将 XML 存放到文件中。

(7) Foreach：针对每个列表项或表格项循环。

(8) Indexof：每个列表项或表格项的索引定位。

(9) Dump：按层存储 swing 元素。

(10) AssertHasFocus：断言被聚焦访问的部件。

(11) AssertEnabled：断言允许访问的部件。

(12) AssertTableCellContains：断言表格项的内容。

(13) AssertTextFieldContains：断言正文区的内容。

3. JFCUnit 特殊语言元素

Click：单击。

Find：发现部件/帧/对话框。

Drag：创建一个拖曳操作。

Record：将 AWT 实践录制到 XML。

4. 用 XMLRoot 进行快速测试

利用 XML 进行录制是一个很简单的应用——只需在 XML 文件中加一个录制标签<record/>即可。输入在录制元素之前放置好，在按 Ctrl+D 键之前，录制一直进行着。在按 Ctrl+D 键后，回放脚本将一直执行到另一个录制标签。下面看一个录制计算器操作的例子

(如图 7-13 所示)。

```
<suite_name="Caculator">
    <!--definition of a local suite -->
    <suite name="Recording test suite">
        <test name="Recording test" robot="true" debug="true">
            <record encoding="UTF-8" file="calculatorSaved.xml"/>
        </test>
    </suite>
</suite>
```

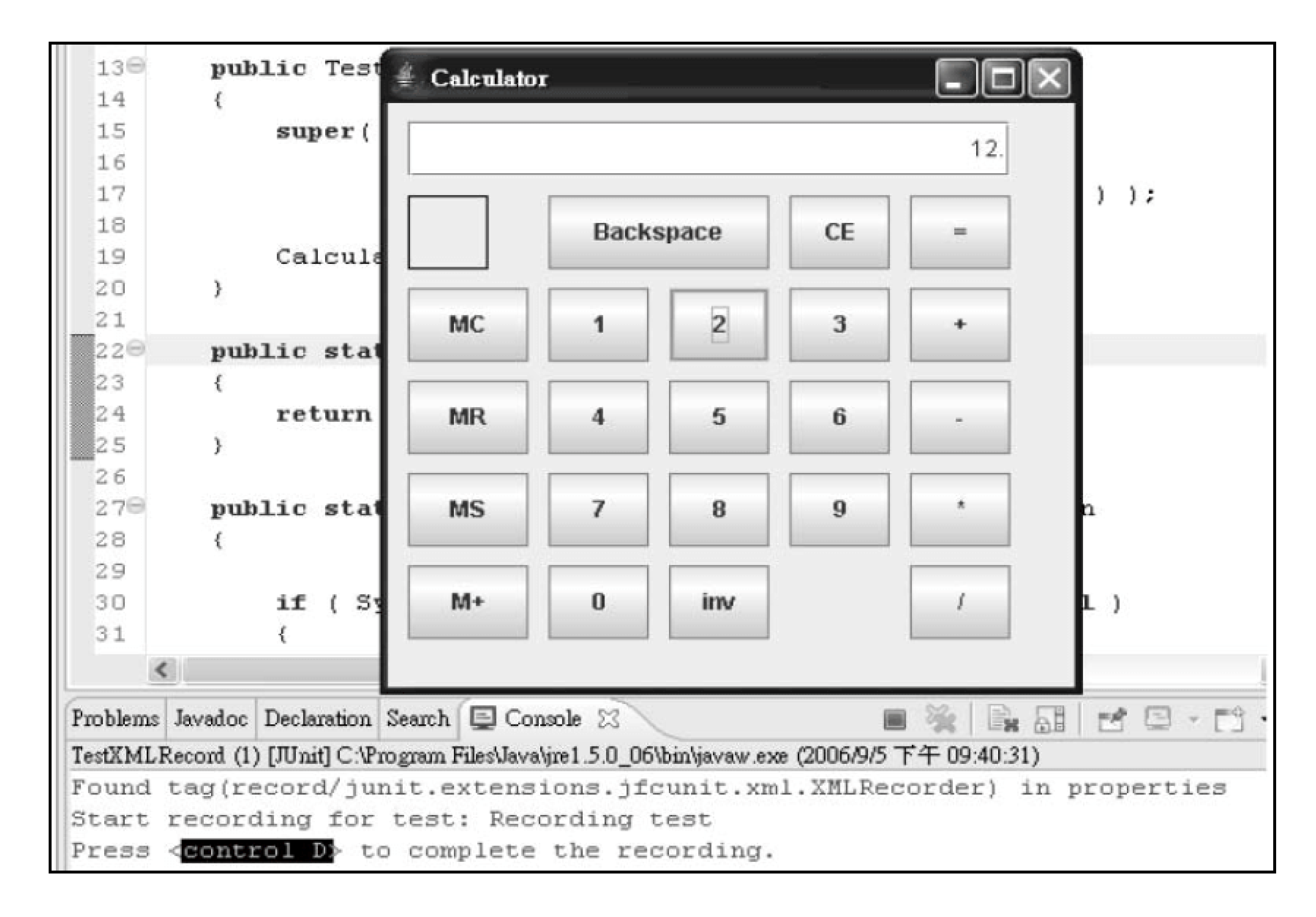


图 7-13　用 XML 对计算器操作进行录制

另外，可以生成一个能够让应用运行起来的普通工具，但该工具覆盖不了所有的环境。junit.extensions.jfcunit.tools.XMLRoot 能够启动应用，启动定义在上述应用上的 XML 中的测试用例。它将生成一个插桩后的 XML 文件，并很快地开始测试用例的生成。它还允许在命令行、系统特性或 API 中使用参数。下面就是一个如何用 SwingSet 快速开始测试的例子。

```
# to start recording to a new testcases.xml
java -Djfcunit.xmlroot.record=true \
     -Djfcunit.xmlroot.classname=demo.SwingSet\
     -Djfcunit.xmlroot.testsuite=testcases.xml\
     -classpath jfcunit.jar;SwingSet.jar;jarkarta-regexp-1.2.jar;junit.jar\
     junit.swingui.TestRunner junit.extensions.jfcunit.tools.XMLRoot
# to re-run recordings
java -Djfcunit.xmlroot.classname=demo.SwingSet\
     -Djfcunit.xmlroot.testsuite=testcases.xml\
```

```
-classpath jfcunit.jar;SwingSet.jar;jarkarta-regexp-1.2.jar;junit.jar\
junit.swingui.TestRunner junit.extensions.jfcunit.tools.XMLRoot
```

注意：当前的 XML 对单个的测试用例有不能独自运行这一限制。

5. Java 编码要求

Java 中保留什么样的代码，如何与 JUnit 测试进行关联并成为由 XML 定义的测试用例？下面给出一段代码说明如何派生一个 XMLTestCase 的代码，并加载和运行 testSwingSet.xml。该测试用例允许将 XMLTestSuite 嵌入到既有 XML 又有 Java 定义的这种很大的自动化测试中。

```
public class TestXMLSwingSet extends XMLTestSuite {
    public static final String DOCUMENT_FACTORY =
         "javax.xml.parsers.DocumentBuilderFactory";
    public TestXMLSwingSet() throws Exception {
        super("testSwingSet.xml",
              XMLUtil.readFileFromClassContext(
                  TestXMLSwingSet.class, "testSwingSet.xml"));
        SwingSet.main(new String[] {});
    }
    public static Test suite() throws Exception {
        return new TestXMLSwingSet();
    }
    public static void main(String[] args) throws Exception {
        if (System.getProperty(DOCUMENT_FACTORY) = = null) {
            System.setProperty(DOCUMENT_FACTORY,
            "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl");
        }
        TestRunner.run((Test) TestXMLSwingSet.suite());
    }
}
```

6. 创建自定义 Tag Handler 的例子

JFCUnit 中已经定义了大量的 Tag Handler，它们几乎包含所有与 Swing 组件相关的数据。但是对于一些特殊要求，也可以自定义 Tag Handler，把它包含到 JFCUnit 中的 taghandler 属性文件中，然后就可以按照 JFCUnit 已定义好的 Tag Handler，以同样方式直接应用到 XML 文件中，扩展图形界面的测试能力。

首先，自定义一个 Tag Handler 类。下面是示意性代码：MyTagHandler.java。

```
package mypackage;
public class MyTagHandler extends junit.extensions.jfcunit.AbstractTagHandler {
```

```
public MyTagHandler (Element element, IXMLTestCase testcase) {
    super (element, testCase);
}
public void validateElement() {
    super. validateElement();
    checkElementTagName("MyTag");
    …
```

上面的 Java 代码用于实现标签 MyTagHandler 的功能，当测试程序从 XML 文件读取这个标签时，就通过此标签对应的程序来执行功能。

其次，对于上面实现的 Tag Handler，将它们添加到 JFCUnit 中已定义的 Tag Handler 文件 TagMapping.properties 中，如下所示：

```
Mytag = mypackage.MyTagHandler;
```

然后就可以在 XML 中使用了，使用方式如下所示：

```
<stopwatch id="MyTag" action="mark"/>
...
<stopwatch refid="MyTag" action="lessthan" value="5000"/>
```

最后，就可以使用这些标记功能反复地测试被测软件的 GUI 图形界面了。如果要发挥 JFCUnit 的功能，就要充分使用它提供的 JFCEventManager 和 API 等。

下面的代码说明了 Tag Handler 的构造方法。AbstractTagHandler 为用户提供了一个处理标签 tag 的工具集。该工具集提供了检索属性值的方法，同时能确定该属性是否存在。

必须提供一个通过参数和 XMLTestCase 来使用 API 的模板，以使相关属性能够与后面要讨论的 TagHandler.properties 文件一起工作。句柄主要是基于 validateElement()和 processElement()这两种方法。validateElement()方法确认在信息中所有必需的信息是可用的。processElement()是实际定义的动作。在下面的例子中，processElement()将一个 ID 标识添加到 IXMLTestCase，检索引用标识并实施与当前时间进行比较的操作。

```
package junit.extensions.xml.elements;
import org.w3c.dom.Element;
import junit.extensions.xml.XMLConstants;
import junit.extensions.xml.IXMLTestCase;
import junit.extensions.xml.XMLException;
/**
* This is tag handler can measure the milliseconds within a test case.
*
* <h3>Description</h3>
* <p>
* Mark the start of the process to be timed with a mark action.
```

```
* <stopwatch action="mark"/>
*
* Then afterward assert that the duration in milliseconds.
* <stopwatch action="lessthan" value="8000"/>
*
*/
public class StopWatchTagHandler
extends AbstractTagHandler
implements XMLConstants {
/**
 * constructor
 * @param element Element to be processed.
 * @param testCase containing test case.
 *
public StopWatchTagHandler(Element element, IXMLTestCase testCase) {
                                super(element, testCase);
 }
/**
 * Validate that the element is properly configured.
 * @throws XMLException Exception may be thrown if there
 * are missing elements.
 */
   public void validateElement() throws XMLException {
     // check the element tag name
     checkElementTagName(STOPWATCH);
     // reqd attribute: at least one of refid or id is present
     checkAtLeastOneRequiredAttribute(getElement(), new String[] {REFID, ID});
     // action is a required attribute
     checkRequiredAttribute(ACTION);
     String action = getString(ACTION);
     if (!action.equals(MARK)) {
     // if action is not a mark, then a value is required.
       checkRequiredAttribute(getElement(), VALUE);
     }
   }
/**
 * Process the element
 */
   public void processElement() {
     String action = getString(ACTION);
     if (action.equals(MARK)) {
```

```
        String id = getString(ID);
        Long mark = new Long(System.currentTimeMillis());
        getXMLTestCase().addFoundObject(id, mark);
      } else {
    //Get the stored objects
        long mark=((Long)getXMLTestCase().getFoundObject(getString("refid"))).longValue();
        long value = getLong(VALUE, 0);
        long newMark = System.currentTimeMillis();
        if (action.equals(LESSTHAN)) {
            getTestCase().assertTrue("Duration exceeded.", newMark - mark >= value);
        }
      }
    }
  }
```

7. 注册 Tag Handlers

现在已经创建了一个 Tag Handler。需要在 XML 文件中注册该句柄，以使测试用例能够找到该句柄并执行其代码。所有的 Tag Handlers 都是全局作用域，可以在测试用例和测试集之间使用。

```
<suite name="my test suite">
  <taghandlers mode="add" name="stopwatch"
            classname="junit.extensions.xml.elements.StopWatchTagHandler"/>
      ...
    </suite>
```

在处理 XML 时，元素名称用于鉴别和执行该 Tag Handler。

8. JFCUnit XML 示例

在 JFCUnit 情况下，提供的许多 Tag Handlers 都能够完成 JFCUnit 所能完成的所有功能。下面是一个由 JFCUnit 定义的测试用例示例，该例说明了一个查找部件的通用模式以及部件上的事件集合。

```
<suite name="JTextComponent">
    <test name="Add my own text" robot="true">
        <manager debug="true" recording="true"/>
        <find finder="ComponentFinder" id="TabPane"
        class="javax.swing.JTabbedPane" index="0"/>
        <click type="JTabbedPaneMouseEventData" refid="TabPane"
        title="Plain Text" clicks="1"/>
        <find finder="ComponentFinder" id="Text"
```

```
        class="javax.swing.text.JTextComponent" index="5"/>
        <click type="JTextComponentMouseEventData" refid="Text" index="0"/>
        <click type="JTextComponentMouseEventData" refid="Text" index="1"/>
        <click type="JTextComponentMouseEventData" refid="Text" index="2"/>
        <click type="JTextComponentMouseEventData" refid="Text" index="3"/>
        <click type="JTextComponentMouseEventData" refid="Text" index="4"/>
        <key refid="Text" code="VK_A" modifiers="control"/>
        <key refid="Text" string="This is the text I wanted to enter." />
    </test>
</suite>
```

9. JFCUnit XML 录制示例

为了录制出一个新的 JFCUnit 测试用例，可以使用下面的模板：

```
<suite name="Suite">
    <test name="Login" robot="true">
        <record/>
    </test>
    <test>
        <save file="new.xml"/>
    </test>
</suite>
```

启动测试用例(可参见前面的 Java 编码需求)。当记录元素与测试用例新元素相匹配时，就在上面加上录制标志。若被保存的标志被处理，当前 XML 文件将被写到指定文件中。随后 new.xml 将包含如下内容：

```
<suite name="Suite">
    <test name="Login" robot="true">
        <find finder="FrameFinder" id="JFrame0" index="0" title="Login"/>
        <find class="javax.swing.JTextField" container="JFrame0"
        finder="ComponentFinder" id="Component1" index="0"/>
        <click clicks="1" index="0" modifiers="16" popup="false" position="0"
        refid="Component1" sleeptime="0" type="JTextComponentMouseEventData"/>
        <key modifiers="0" position="0" refid="Component1" sleeptime="0" string="froto"/>
        <find class="javax.swing.JPasswordField" container="JFrame0"
        finder="NamedComponentFinder" id="Component2" index="0"
        name="Password text field"/>
        <click clicks="1" index="0" modifiers="16" popup="false" position="0"
        refid="Component2" sleeptime="0" type="JTextComponentMouseEventData"/>
        <key modifiers="0" position="0" refid="Component2" sleeptime="0"
        string="baggins"/>
```

```
        <find class="javax.swing.JTextField" container="JFrame0"
        finder="ComponentFinder" id="Component3" index="2"/>
        <click clicks="1" index="0" modifiers="16" popup="false" position="0"
        refid="Component3" sleeptime="0" type="JTextComponentMouseEventData"/>
        <key modifiers="0" position="0" refid="Component3" sleeptime="0"
        string="localhost"/>
        <find container="JFrame0" finder="AbstractButtonFinder" id="Component4"
        index="0" label="&gt;&lt;Login"/>
        <click clicks="1" modifiers="16" popup="false" position="12" reference="22,5"
        refid="Component4" sleeptime="0" type="MouseEventData"/>
        <record file="new.xml"/>
    </test>
  </suite>
```

10. JFCUnit XML 方法总结

首先建立一个测试包框架，然后编写测试 XML 文件。其过程概述如下。

(1) 创建一个 XML 测试类，程序框架参见前面“Java 编码要求”中的代码。

(2) 使用默认的 Tag Handler 特性文件中的 Tag Handler 编写出各种测试集和具体的测试用例。上层的 Java 类会读取这些标签句柄来进行测试。JFCUnit XML 文件可参见前面的“JFCUnit XML 示例”。建立好测试框架程序后，只需编写 XML 文件中的测试事件和序列即可完成 GUI 图形界面的测试。

(3) 自定义标签句柄(Tag Handler)，扩展 JFCUnit 的测试能力。

实 验 习 题

1. 对 JFCUnit 包中的例子进行测试，并完成前面售货机程序的界面测试，给出测试结果。

2. 利用 JFCUnit 工具，对一个 XML 实例进行测试，给出测试的流程及结果。

第8章　Web页面测试

随着 Internet 和 Intranet 的快速发展，Web 技术已经对工商业、医疗业、教育、政府、娱乐以及人们的生活产生了深远的影响。Web 平台能支持几乎所有媒体类型的信息发布，容易为最终用户存取，更多传统的信息和数据系统正在逐渐迁移到互联网上：电子商务正迅速增长，范围广泛、复杂的云应用和云计算也正在 Web 环境中出现。基于 Web 的系统在变得越来越复杂和强大的同时，Web 应用软件的缺陷危机也越来越严重。早在 1998 年 Yogesh Deshpande 和 Steve Hansen 就提出了 Web 工程的概念。Web 工程提倡使用一个过程和系统的方法来开发高质量的基于 Web 的系统。在 Web 工程中，基于 Web 系统的测试、确认和验收是一项重要而富有挑战性的工作。

Web 环境具有浏览器平台不兼容、网络环境多样化、应用复杂化等诸多特性，所以，传统测试方法的某些方面不适用于网络测试。Web 的自动化测试方法包含几个方面，比如，测试脚本技术、人工测试过程自动化、验证自动化等。在测试驱动开发模式中，测试已成为迭代开发过程中起推动作用的环节，但与此同时，大量的重复性的测试代码却造成了大量资源的浪费。

随着自动化测试技术的成熟和自动化测试工具的广泛应用，人们重新认识到了测试的价值：最优的质量成本，最高的质量保证。自动化测试的优势在软件领域是很明显的：减少了测试时间，使测试程序统一化，便于管理，节约了质量保证的成本，提高了测试运行的效率，改善了软件产品的质量。

现在一般人都有使用浏览器浏览网页的经历，用户虽然不是专业人员但是对界面效果的印象是很重要的。如果开发人员注重这方面的测试，那么验证应用程序是否易于使用就非常重要了。很多人认为这是测试中最不重要的部分，但是恰恰相反，界面对不懂技术的客户来说都是相当关键的，特别是在简洁、美观、易用等方面。

方法上可以根据设计文档，如果够专业可以由专业美工人员来确定整体风格，特别是页面风格。然后根据这个设计好的页面，生成静态的 HTML、CSS 等甚至生成几套不同的方案来讨论，或者交给客户评审，最后形成统一风格的页面/框架。

页面测试的主要页面元素如下。

(1) 页面元素的容错性列表(如输入框、时间列表或日历)。

(2) 页面元素清单(为实现功能，是否将所需要的元素全部都列出来了，如按钮、单选按钮、复选框、列表框、超链接、输入框等)。

(3) 页面元素的容错性是否存在。

(4) 页面元素的容错性是否正确。

(5) 页面元素的基本功能是否实现(如文字特效、动画特效、按钮、超链接)。

(6) 页面元素的外形、摆放位置(如按钮、列表框、复选框、输入框、超链接等)。

(7) 页面元素是否显示正确(主要针对文字、图形、签章)。

(8) 元素是否显示(元素是否存在)。

页面测试主要包括以下几个方面的内容。

(1) 站点地图和导航条位置是否合理，是否可以导航等。

(2) 页面内容布局是否合理，文字是否准确、简洁，字体和字号是否符合多数读者习惯。

(3) 背景/色调是否合理、美观，是否符合多数用户的审美要求。

(4) 页面在窗口中的显示是否正确、美观(在调整浏览器窗口大小时，屏幕刷新是否正确)，表单样式大小、格式是否适宜。

(5) 是否对提交数据进行验证(如果在页面部分进行验证)等。

(6) 链接的形式、位置、是否易于理解等。

对 Web 应用的测试可以分为页内测试(IntraPageTest)和跨页测试(InterPageTest)两种。页内测试相当于单元测试，着重于测试单个页面的行为是否正确。根据模块化思想，在进行页面划分时，一般使每个页面具有单一、具体的功能，可以直接表达用户的一个目标。本章主要考虑 Web 页内测试的主要方法。

(1) 人工走查：①通过页面走查，浏览确定使用的页面是否符合需求。可以结合兼容性测试不同分辨率下的页面显示效果，如果有影响，则应该交给设计人员由他们提出解决方案。②可以结合数据定义文档查看表单项的内容、长度等信息。③对于动态生成的页面最好也能浏览查看。如 Servlet 部分可以结合编码规范，进行代码走查。是否支持中文，如果数据用 XML 封装，要做的工作可能会多一点。

(2) 使用 Web 页面测试工具：工具可以提供很多测试方法，可以用来模拟许多手工操作，如单击按钮、给文本框输入字符或数字、鼠标双击事件等，从而实现了测试的自动化。这对于需要输入大量信息的界面测试来说是十分重要的。

Web 页面测试的基本准则：符合页面/界面设计的标准和规范，满足灵活性、正确性、直观性、舒适性、实用性、一致性等要求。

直观性：①用户界面是否洁净，不唐突、不拥挤，界面不应该为用户制造障碍，所需功能或者期待的响应应该明显，并在预期的地方出现。②界面组织和布局合理吗?是否允许用户轻松地从一个功能转到另一个功能?下一步做什么明显吗?任何时刻都可以决定放弃或者退回、退出吗?输入得到承认了吗?菜单或者窗口是否深藏不露?③有多余功能吗?软件整体抑或局部是否做得太多?是否因有太多特性而把工作复杂化了?是否感到信息太庞杂?④如果其他所有努力失败，帮助系统真能帮忙吗?

一致性：①快捷键和菜单选项，在 Windows 中按 F1 键总是得到帮助信息。②术语和命令，整个软件使用同样的术语吗?特性命名一致吗?例如，Find 是否一直叫 Find，而不是有时叫 Search? ③软件是否一直面向同一级别用户?带有花哨用户界面的趣味贺卡程序不应该显示泄

露技术机密的错误提示信息。④按钮位置和等价的按键。读者是否注意到对话框有 OK 按钮和 Cancel 按钮时，OK 按钮总是在上方或者左方，而 Cancel 按钮总是在下方或右方?同样原因，Cancel 按钮的等价按键通常是 Esc，而 OK 按钮的等价按钮通常是 Enter，要保持一致。

灵活性：①状态跳转，灵活的软件实现同一任务时通常会有多种选择方式。②状态终止和跳过，具有容错处理能力。③数据输入和输出，用户希望有多种方法输入数据和查看结果。例如，要在写字板中插入文字，可用键盘输入、粘贴、从 6 种文件格式读入、作为对象插入，或者用鼠标从其他程序拖动。

舒适性：①恰当，软件外观和感觉应该与所做的工作和使用者相符。②错误处理，程序应该在用户执行严重错误的操作之前提出警告，并允许用户恢复由于错误操作导致丢失的数据。正如人们认为 undo /redo 功能是理所当然应有的。③性能，快不见得是好事，要让用户看清程序在做什么，它是有反应的。

8.1 Web 页面测试工具介绍

基于 Web 的测试基本上采用两种思路和方法。一种可以称为“浏览器测试”，这种测试通常是模拟浏览器端的一些操作，比如在 TextBox 中输入一些文本，选择表单组件中的某个部件等。因为可以看见具体的操作界面，这种方法更多地被应用到 UI 和本地化方面的测试中。

另一种方法称为“协议测试”。这是建立在 HTTP 之上的 JavaScript 级别的测试，通过 JavaScript 伪协议或者 POST、Web Service 向服务器发送请求，然后对服务器响应回来的数据进行解析、验证。一些功能测试会更多地采用这种方法。最简单的应用就是检查链接的有效性：向服务器发送 URL 请求，检查响应回来的数据，来判断链接是否指向了正确的页面。

本章所介绍的 Web 页面测试工具事实上能够很好地支持上述测试方法，并且它们均是以 JUnit 为基础扩展出来的，但应用模式各有不同。

1. HttpUnit

HttpUnit 是在 JUnit 之上构建的测试框架，它支持 Web 应用的黑盒测试和 in-container 容器内测试。作为功能测试工具，可以用它来验证软件是否符合业务需求，以及是否在可视的级别符合预期行为。有趣的是，HttpUnit 的基础代码实际上与测试没什么关系。开发 HttpUnit 库的目的是加强 HTTP 对 Web 应用的访问，它支持的特征包括状态管理(cookies)、请求提交、应答解析(HTML 解析)，以及网络爬虫(Web Spider)工具包需要的一些特征。HttpUnit 还有一个支持容器内测试的类 ServletUnit。在 JUnit 提供的断言功能和结果报告功能的基础上，HttpUnit 成了一个非常有用的测试 Web 应用的工具。可以在 www.httpunit.org 上找到 HttpUnit。

2. JWebUnit

JWebUnit 是在 HttpUnit 上创建的一个辅助工具包，它减少了测试 Web 程序所需要编写的代码。简单地说，可以把它当作 HttpUnit 的宏程序库，提供到 HttpUnit 代码段的快捷

方式，简化 Web 程序测试中的大多数行为。HttpUnit 提供的相对底层的接口可以让测试人员定制许多事情。如果用 HttpUnit 可以解决问题，那么用 JWebUnit 也可以。JWebUnit 的好处是它对代码有更好的控制。可以在 http://sourceforge.net/projects/jwebunit 上找到 JWebUnit。

3. StrutsTestCase

StrutsTestCase 是为测试 Struts 应用而在 JUnit 基础上创建的测试框架。Struts 是用 Java 开发 Web 应用的程序员非常喜欢的模型-视图-控制器(MVC)平台，它简化了数据、表示和逻辑分离的易维护性组件式代码开发。Struts 使 Web 程序容器间(in-container)的功能测试和单元测试变得更复杂了，因为它们夹在 Servlet 容器和程序之间。这就意味着这个测试框架要认识 Struts，能处理 Struts 的容器间测试。由于不需要知道 Web 程序的内部实现，所以尽管 HttpUnit 的黑盒测试仍然工作得很好，也仍然无法用 HttpUnit 做 Struts 应用的容器间测试，因为 HttpUnit 是独立地位于程序和 Servlet 容器之间。StrutsTestCase 是专为 Struts 程序的容器间测试而设计的。StrutsTestCase 可以从 http://sourceforge.net/projects/strutstestcase 获得。

4. Selenium

Selenium 是一个用于 Web 应用程序测试的工具，它将核心组件内插到浏览器中。Selenium 测试直接运行在浏览器中，就像真正的用户在操作一样，所有的测试都是可见的。支持的浏览器包括 IE、Mozilla、Firefox 等。这个工具的主要功能包括：测试与浏览器的兼容性，即测试应用程序是否能够很好地工作在不同浏览器和操作系统之上，测试系统功能，检验软件功能和用户需求。Selenium 可从 http://seleniumhq.org/download/上下载。

5. HtmlUnit

HtmlUnit 是一个具有浏览器基本功能的 Java 组件。HtmlUnit 是 JUnit 的扩展测试框架之一，它支持 HTML 文件，并提供了一些 API，允许访问网页、填写表格、单击链接等。HtmlUnit 将返回文档模拟成 HTML，这样便可以直接处理这些文档了。HtmlUnit 使用诸如 table、form 等标识符将测试文档作为 HTML 来处理。它同样需要遵循 JUnit 测试框架结构的 Java 测试程序。这里要注意的是：前面介绍的 HttpUnit 主要是在 HTTP 的 Request 和 Response 层次上进行操作，而 HtmlUnit 则是在比 HTTP 高一些的 HTML 层次上进行操作。HtmlUnit 可从 http://sourceforge.net/projects/htmlunit 上下载。

8.2　Web 页面测试工具之一——HttpUnit

HttpUnit 是 SourceForge 下面的一个开源项目，它是基于 JUnit 的一个测试框架，是一个集成测试工具，主要关注于测试 Web 应用，提供的帮助类让测试者可以通过 Java 类和服务器进行交互，并且将服务器端的响应当作文本或 DOM 对象进行处理，解决使用 JUnit 框架无法对远程 Web 内容进行测试的弊端。HttpUnit 还提供了一个模拟 Servlet 容器，让

用户可以不需要发布 Servlet，就可以对 Servlet 的内部代码进行测试。当前的最新版本是 1.7。为了使 HtpUnit 能正常运行，需要安装 Java JDK 1.3.1 或以上版本。

1. HttpUnit 的工作原理

HttpUnit 不需要使用浏览器。可以使用 HttpUnit 直接调用要测试的代码。从本质上来说，HttpUnit 是模拟 Web 浏览器，并且 HttpUnit API 可以模拟浏览器的许多行为，包括表单提交、JavaScript、HTTP 认证和 Cookie 等。也可以在装入 Web 页面时用 HttpUnit API 分析返回的内容。

HttpUnit 通过模拟浏览器的行为，处理页面帧(frames)、Cookies、页面重定向(redirects)等。通过 HttpUnit 提供的功能，可以和服务器端进行信息交互，将返回的网页内容作为普通文本、XML DOM 对象或者是作为链接、页面框架、图像、表单、表格等的集合进行处理，然后使用 JUnit 框架进行测试；还可以导向一个新的页面，然后进行新页面的处理，这个功能使用户可以处理一组在一个操作链中的页面，轻松地测试 Web 页面。

HttpUnit 可以被分为以下两个核心组件。

(1) 一个发送请求并接收响应的 Web 客户机。

(2) 一个分析并验证响应内容的方法集。

在这里要澄清一个概念——使用 HttpUnit 是不是相当于进行单元测试？

与其名称可能暗示的相反，HttpUnit 实际上并不做单元测试。实际上，HttpUnit 更适合做功能测试或“黑盒”测试。测试人员用 HttpUnit 编写的测试从外部查询 Web 服务器并使用户能够分析接收到的响应。功能测试在 XP 方法中起着重要的作用，这种方法强调功能测试，使开发者可以获取有关系统整体状态的反馈。使用单元测试，有时会丢失大方向。在将整个站点投入到实际使用的过程中，自动功能测试可以使开发者从手工检查站点区域的繁重工作中解脱出来。在 Web 环境中，有些专家认为每个“请求-响应”周期都是原子的，这些原子的周期可以依次被作为单独的代码单元，因此使用 HttpUnit 可以被认为是在进行这种意义上的单元测试。

2. HttpUnit 和其他商业工具的对比

商业工具一般使用录制、回放功能来实现测试，但是这里有个缺陷，就是当页面设计被修改以后，这些被记录的行为就不能重用了，需要重新录制才能继续测试。

举个例子：如果页面上有个元素最先的设计是采用单选框，这个时候开始测试，那么这些工具记录的就是单项选择动作；但是如果设计发生了变化，比如改成了下拉选择，或者使用文本框来接收用户输入，那么这时候，以前录制的测试过程就无效了，必须重新录制。

而 HttpUnit 因为关注点是这些控件的内容，所以不管控件的外在表现形式如何变化，都不会影响已确定测试的可重用性。

最重要的是，由于 Web 应用的快速发展以及 Web 编程错误容易产生的特征，市场上出现的许多商业测试产品都用详细的 GUI 来引导开发者进行测试。另一方面，开放源代码的 HttpUnit 无须许可费用，而且使用简单(将在下面看到这一点)。正是它的简单性，使它如此受欢迎。虽然没有漂亮的 GUI，但 HttpUnit Java 源代码的可用性和它的简单 API，使

开发者能够创建他们自己的测试解决方案以满足特定组织的需要。以 HttpUnit 作为基础，可以用最小的成本构建复杂的测试套件。

8.2.1　HttpUnit 环境建立

可以从 HttpUnit 项目网站 http://www.httpunit.org 上下载到最新版的 HttpUnit，HttpUnit 应运行在支持 Java JDK 1.3 及更高版本的系统上。

将 HttpUnit 解压缩到 c:/httpunit(后面将使用“%httpunit_home%”引用该目录)，目录结构应该如下所示：

```
httpunit
    +--- jars //包含创建、测试及运行 HttpUnit 所必需的 jar
    |
    +--- lib   //  包含 HttpUnit jar
    |
    +--- doc   //文档
    |       |
    |       +--- tutorial   //基于 Servlet Web 网站的测试优先开发的简单教程
    |       |
    |       +--- api          // javadoc
    |       |
    |       +--- manual      //  用户手册
    |
    +--- examples //  采用 HttpUnit 编写的一些示例程序
    |
    +--- src         // HttpUnit  源代码
    |
    +--- test        // HttpUnit 单元测试的一些很好的例子
```

只有 lib 和 jars 这两个目录对于运行 HttpUnit 来说是必需的。必须将 HttpUnit jar 添加到系统的 classpath，而其他的一些 jar 均为可选。

由于考虑 Web 页面是在 Eclipse 中开发、执行的，所以环境配置都是以 Eclipse 为例。如果使用其他的开发工具，则可根据以下步骤进行环境配置。

(1) 启动 Eclipse，建立一个 Java 项目。

(2) 将%httpunit_home%/lib/*.jar、%httpunit_home%/jars/*.jar 加入到该 Java 项目的 Java build Path 变量中。

8.2.2　HttpUnit 的工作方式

1. HttpUnit 重点类的应用

在使用 HttpUnit 进行页面测试时，需要特别关注如下几个类。

(1) WebConversation 是 HttpUnit 框架中最重要的类，它用于模拟浏览器的行为。

(2) WebRequest 模仿客户请求，通过它可以向服务器发送信息。

(3) WebResponse 模拟浏览器，获取服务器端的响应信息。

下面通过示例介绍上述类的具体应用。

1) 类的最基本应用

```
System.out.println("直接获取网页内容: ");
// 建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
// 向指定的 URL 发出请求，获取响应
WebResponse wr = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
// 用 getText 方法获取响应的全部内容
// 用 System.out.println 将获取的内容打印在控制台上
System.out.println( wr.getText() );
```

2) 通过 Get 方法访问页面并加入参数

```
System.out.println("向服务器发送数据，然后获取网页内容: ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//向指定的 URL 发出请求
WebRequest req = new GetMethodWebRequest(" http://localhost:6888/HelloWorld.jsp " );
//给请求加上参数
req.setParameter("username","姓名");
//获取响应对象
WebResponse resp = wc.getResponse(req);
//用 getText 方法获取响应的全部内容
//用 System.out.println 将获取的内容打印在控制台上
System.out.println(resp.getText());
```

3) 通过 Post 方法访问页面并加入参数

```
System.out.println("使用 Post 方式向服务器发送数据，然后获取网页内容: ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//向指定的 URL 发出请求
WebRequest req = new PostMethodWebRequest(" http://localhost:6888/HelloWorld.jsp ");
//给请求加上参数
req.setParameter("username","姓名");
//获取响应对象
WebResponse resp = wc.getResponse(req);
//用 getText 方法获取响应的全部内容
//用 System.out.println 将获取的内容打印在控制台上
```

```
System.out.println(resp.getText());
```

读者可以关注上面代码中打了下划线的两处内容，应该可以看到，使用 Get、Post 方法访问页面的区别就是使用的请求对象不同。

2. 处理页面中的链接

找到页面中的某一个链接，然后模拟用户的单击行为，获得它指向文件的内容。比如在页面 HelloWorld.html 中有一个链接，它显示的内容是 TestLink，它指向另一个页面 TestLink.html。TestLink.html 页面只显示 TestLink.html 页面的几个字符。

下面是处理代码：

```
System.out.println("获取页面中链接指向页面的内容：");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
//获得页面链接对象
WebLink link = resp.getLinkWith( "TestLink" );
//模拟用户单击事件
link.click();
//获得当前的响应对象
WebResponse nextLink = wc.getCurrentPage();
//用 getText 方法获取响应的全部内容
//用 System.out.println 将获取的内容打印在控制台上
System.out.println( nextLink.getText() );
```

3. 处理页面中的表格

表格是用来控制页面显示的常规对象，HttpUnit 使用数组来处理页面中的多个表格，可以用 resp.getTables()方法获取页面中所有的表格对象。它们依照出现在页面中的顺序保存在一个数组里面。

注意：Java 中数组的下标是从 0 开始的，所以取第一个表格时应该是 resp.getTables()[0]，其他以此类推。

下面的示例演示了如何从页面中取出第一个表格的内容并将它们循环显示出来。

```
System.out.println("获取页面中表格的内容：");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
//获得对应的表格对象
WebTable webTable = resp.getTables()[0];
```

```
//将表格对象的内容传递给字符串数组
String[][] datas = webTable.asText();
//循环显示表格内容
int i = 0 ,j = 0;
int m = datas[0].length;
int n = datas.length;
while (i<n){
    j=0;
    while(j<m){
        System.out.println("表格中第"+(i+1)+"行第"+(j+1)+"列的内容是： "+datas[j]);
        ++j;
    }
    ++i;
}
```

4. 处理页面中的表单

表单用来接收用户输入，也可以向用户显示用户已输入的信息(如需要用户修改数据时，通常会显示他之前输入过的信息)。HttpUnit 使用数组来处理页面中的多个表单，可以用 resp.getForms()方法获取页面中所有的表单对象。它们依照出现在页面中的顺序保存在一个数组里面。

注意：Java 中数组的下标是从 0 开始的，所以取第一个表单时应该是 resp.getForms()[0]，其他以此类推。

下面的示例演示了如何从页面中取出第一个表单的内容并将它们循环显示出来。

```
System.out.println("获取页面中表单的内容： ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/HelloWorld.html " );
//获得对应的表单对象
WebForm webForm = resp.getForms()[0];
//获得表单中所有控件的名称
String[] pNames = webForm.getParameterNames();
int i = 0;
int m = pNames.length;
//循环显示表单中所有控件的内容
while(i<m){
    System.out.println("第"+(i+1)+"个控件的名称是"+pNames+"，\
                        里面的内容是"+webForm.getParameterValue(pNames));
    ++i;
}
```

5. 如何使用 HttpUnit 进行测试

1) 对页面内容进行测试

HttpUnit 本身没有测试功能，事实上，它就是由一些类库组成，可以模拟出一个浏览器(WebConversation 类)，并可以模拟用户在网页上的多种行为。HttpUnit 要实现 Web 测试功能，需要结合 JUnit 才行。所以，HttpUnit 中的这部分测试完全采用了 JUnit 的测试方法，即直接将期望的结果和页面中的输出内容进行比较。这里只是字符串和字符串的比较。比如，期望的页面显示是其中有一个表格，并且是页面中的第一个表格，而且它的第一行第一列显示的数据应该是 username；那么，可以使用下面的代码进行自动化测试。

```
System.out.println("获取页面中表格的内容并且进行测试: ");
//建立一个 WebConversation 实例
WebConversation wc = new WebConversation();
//获取响应对象
WebResponse resp = wc.getResponse( " http://localhost:6888/TableTest.html " );
//获得对应的表格对象
WebTable webTable = resp.getTables()[0];
//将表格对象的内容传递给字符串数组
String[][] datas = webTable.asText();
//对表格内容进行测试
String expect = "中文";
Assert.assertEquals(expect,datas[0][0]);
```

2) 对 Servlet 进行测试

除了对页面内容进行测试外，有时候(比如开发复杂的 Servlet 的时候)，还需要对 Servlet 本身的代码块进行测试。可以选择 HttpUnit，它可以提供一个模拟的 Servlet 容器，让 Servlet 代码不需要发布到 Servlet 容器(如 Tomcat)就可以直接测试。

使用 HttpUnit 测试 Servlet 时，需要创建一个 ServletRunner 实例，它负责模拟 Servlet 容器环境。如果只是测试一个 Servlet，那么可以直接使用 registerServlet 方法注册这个 Servlet；如果需要配置多个 Servlet，那么可以编写自己的 web.xml，然后在初始化 ServletRunner 的时候将它的位置作为参数传给 ServletRunner 的构造器。

在测试 Servlet 时，应该记得使用 ServletUnitClient 类作为客户端，它和前面用过的 WebConversation 差不多，都继承自 WebClient，所以它们的调用方式基本一致。要注意的是，在使用 ServletUnitClient 时，它会忽略 URL 中的主机地址信息，而是直接指向 ServletRunner 所实现的模拟环境。

下面的示例只是演示了如何简单地访问 Servlet 并获取它的输出信息。例子中 Servlet 在接到用户请求的时候只是返回一串简单的字符串“Hello World!”。

Servlet 的代码如下：

```
public class MyServlet extends HttpServlet {
```

```
public void service(HttpServletRequest req, HttpServletResponse resp)
throws IOException
{
    PrintWriter out = resp.getWriter();
    //向浏览器中写一个字符串 Hello World!
    out.println("<html><body>Hello World!</body></html>");
    out.close();
  }
}
```

测试的调用代码如下：

```
//创建 Servlet 的运行环境
ServletRunner sr = new ServletRunner();
//向环境中注册 Servlet
sr.registerServlet( "myServlet", MyServlet.class.getName() );
//创建访问 Servlet 的客户端
ServletUnitClient sc = sr.newClient();
//发送请求
WebRequest request = new GetMethodWebRequest( " http://localhost/myServlet " );
//获得模拟服务器的信息
WebResponse response = sc.getResponse( request );
//将获得的结果打印在控制台上
System.out.println(response.getText());
```

测试 Servlet 的内部行为：

对于开发者来说，仅测试请求和返回信息是不够的，所以 HttpUnit 提供的 ServletRunner 模拟器可以对被调用 Servlet 的内部行为进行测试。和简单测试中不同，这里使用了 InvocationContext 来获得该 Servlet 的环境，然后可以通过 InvocationContext 对象针对 request、response 等对象或者该 Servlet 的内部行为(非服务方法)进行操作。

下面的代码演示了如何使用 HttpUnit 模拟 Servlet 容器，并通过 InvocationContext 对象测试 Servlet 内部行为的大部分工作，比如控制 request、session、response 等。

```
//创建 Servlet 的运行环境
ServletRunner sr = new ServletRunner();
//向环境中注册 Servlet
sr.registerServlet( "InternalServlet", InternalServlet.class.getName() );
//创建访问 Servlet 的客户端
ServletUnitClient sc = sr.newClient();
//发送请求
WebRequest request = new GetMethodWebRequest( " http://localhost/InternalServlet " );
request.setParameter("pwd","pwd");
```

```
//获得该请求的上下文环境
InvocationContext ic = sc.newInvocation( request );
//调用 Servlet 的非服务方法
InternalServlet is = (InternalServlet)ic.getServlet();
is.myMethod();
//直接通过上下文获得 request 对象
System.out.println("request 中获取的内容: "+ic.getRequest().getParameter("pwd"));
//直接通过上下文获得 response 对象,并向客户端输出信息
ic.getResponse().getWriter().write("haha");
//直接通过上下文获得 session 对象，控制 session 对象
//给 session 赋值
ic.getRequest().getSession().setAttribute("username","timeson");
//获取 session 的值
System.out.println("session 中的值: "+ic.getRequest().getSession().getAttribute("username"));
//使用客户端获取返回信息，并且打印出来
WebResponse response = ic.getServletResponse();
System.out.println(response.getText());
```

注意在测试 Servlet 之前，必须通过 InvocationContext 完成 Servlet 中的 service 方法中完成的工作；因为通过 newInvocation 方法获取 InvocationContext 实例的时候，该方法并没有被调用。

8.3　Web 页面测试工具之二——JWebUnit

JWebUnit 是基于 Java 的用于测试网络程序的框架，架构在 HttpUnit 之上——即 JWebUnit 以HttpUnit和 JUnit单元测试框架为基础，适合做 Web 应用的验收测试。实际上也可以这么说，JWebUnit 是 HttpUnit 的更高一层 API 封装，提供了访问 Web 应用程序的高级 API，并组合了一组断言，用它们来验证链接导航、表单输入项和提交、表格内容以及其他典型业务类 Web 应用程序特性的正确性，避免了使用 HttpUnit 来编写烦琐复杂的测试用例。JWebUnit 是以 JAR 文件形式提供的，可以很容易地将它作为插件集成到大多数的 IDE 中，JWebUnit 还包含其他必要的库。

除了底层的一些逻辑 API 外，JWebUnit 还轻量地集成了现有的测试框架 HtmlUnit 和 Selenium，并提供统一的一组简单易用的接口，可以很方便地与 HtmlUnit 或 Selenium 测试项目集成。所以对于目标测试用例来说，所有的接口都是透明且统一语义的，这将极大保证测试用例的通用性并降低了开发难度。

8.3.1　JWebUnit 测试环境建立

JWebUnit 可以通过单击链接 http://sourceforge.net/projects/jwebunit 来下载。JWebUnit 编译需要下面这几个包的支持：HttpUnit Java 库包，HttpUnit 用到的 Rhino JavaScript(仅在对 JavaScript 进行测试时需要)，HttpUnit 用到的 nekohtml 分析器和美化器，JUnit。另外，如用到 Tidy/HttpUnit 或 HttpUnit/neko 等功能，则还需下载 Apache XML API Common XML stuff 和 Xerces xml parser 等。

下载完 JWebUnit 之后，请按以下步骤在 Eclipse 平台上配置 JWebUnit 库。

(1) 把下载的 JWebUnit 文件释放到临时目录中(假设是 C:\temp)。JWebUnit 当前的最新版本为 3.1。

(2) 在 Eclipse(这里用到的测试环境为 MyEclipse 6.0.1)中创建新 Java 项目，将其命名为 JWebUnitTest。

(3) 右击 Package Explorer 视图中的 JWebUnitTest 项目，然后选择 Properties 命令。

(4) 单击 Java Build Path。单击 Libraries 选项卡中的 Add External JARs 按钮。

(5) 浏览到 C:\temp\jwebunit-3.1\lib 目录，选择这个目录中的所有 JAR 文件。

(6) 单击 OK 按钮。

现在可以在 Eclipse 中的 JWebUnitTest 项目下开发 JWebUnit 测试用例了，如图 8-1 所示。

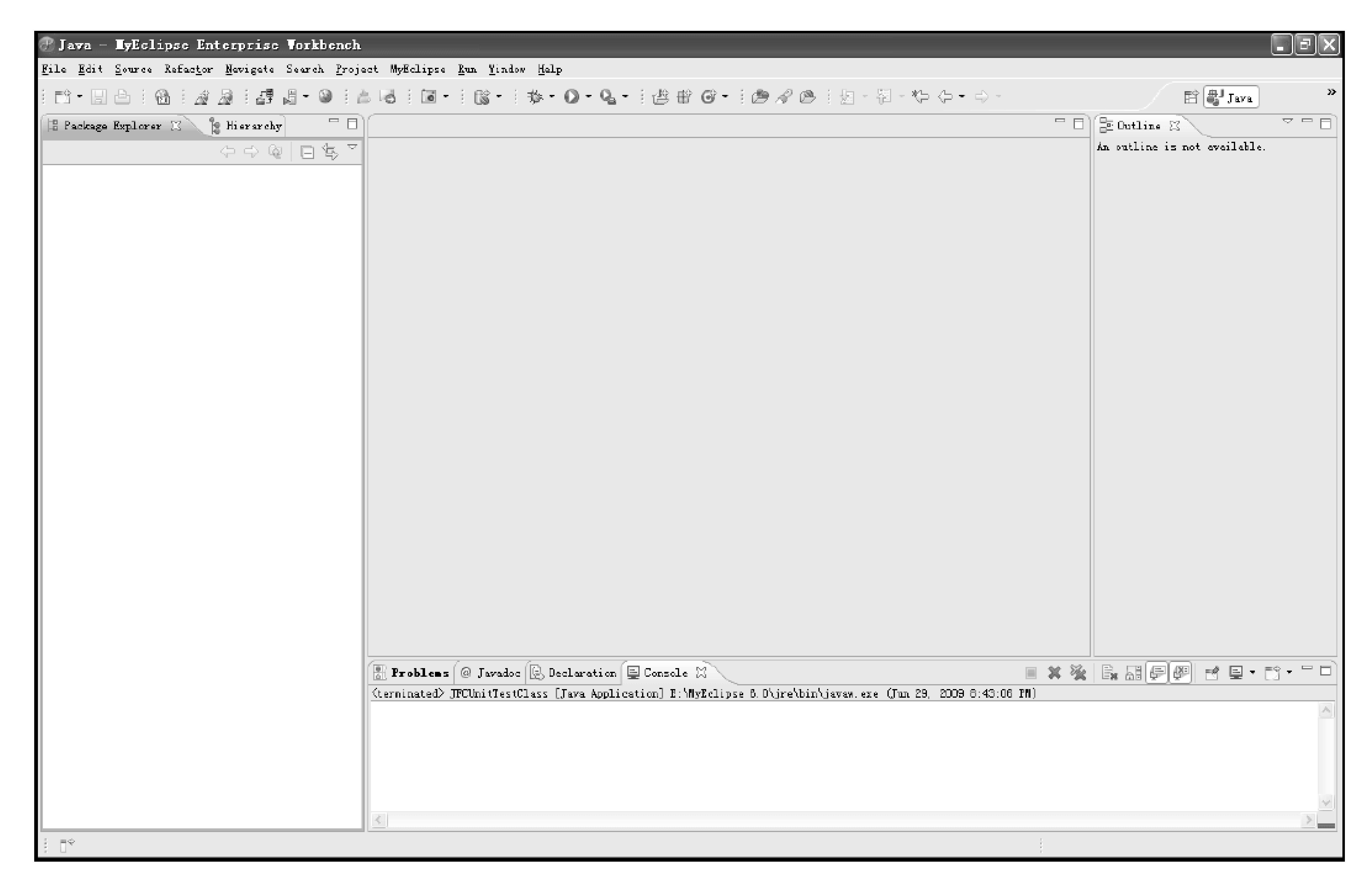
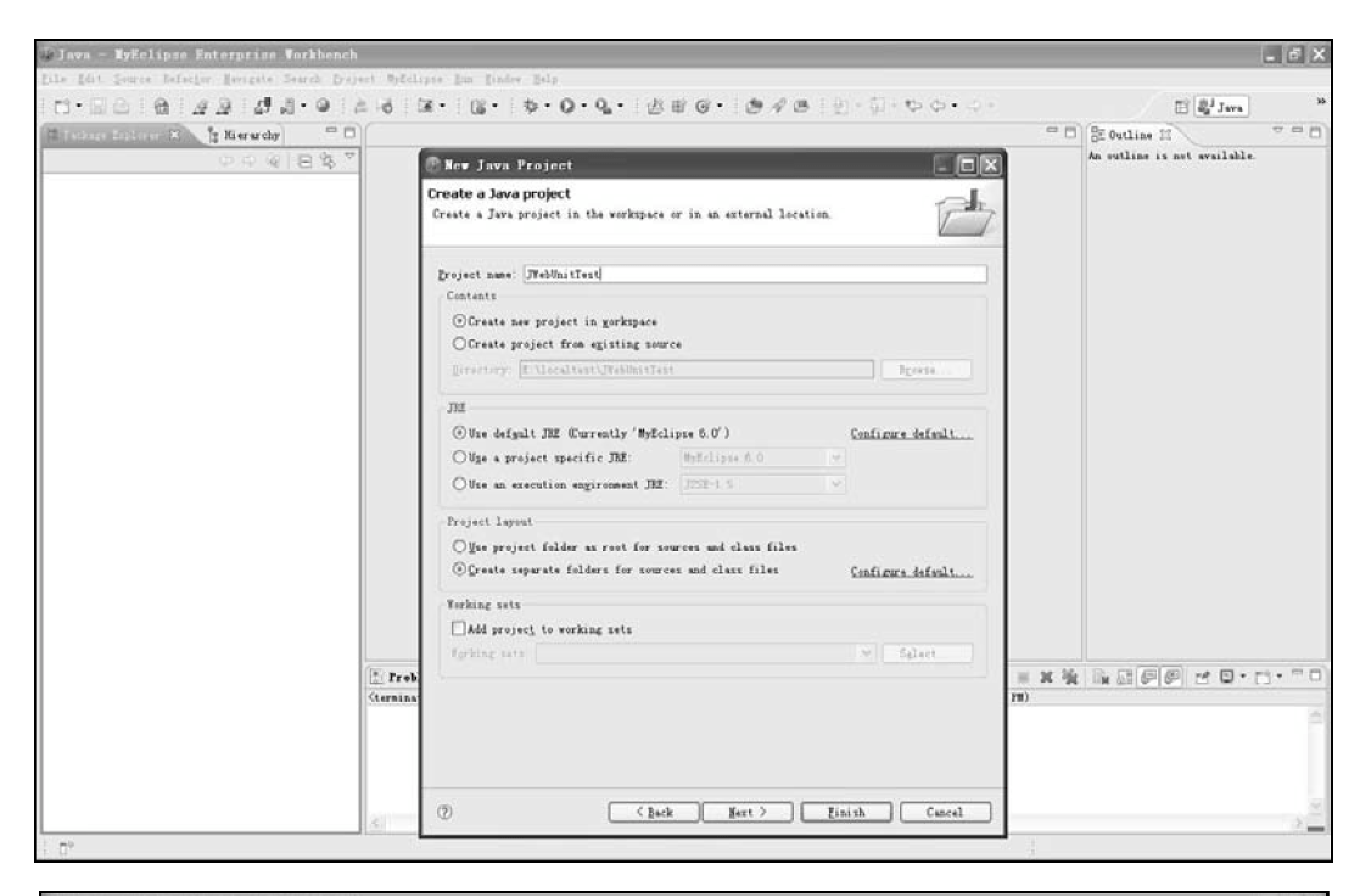


图 8-1　建立 JWebUnit 测试环境



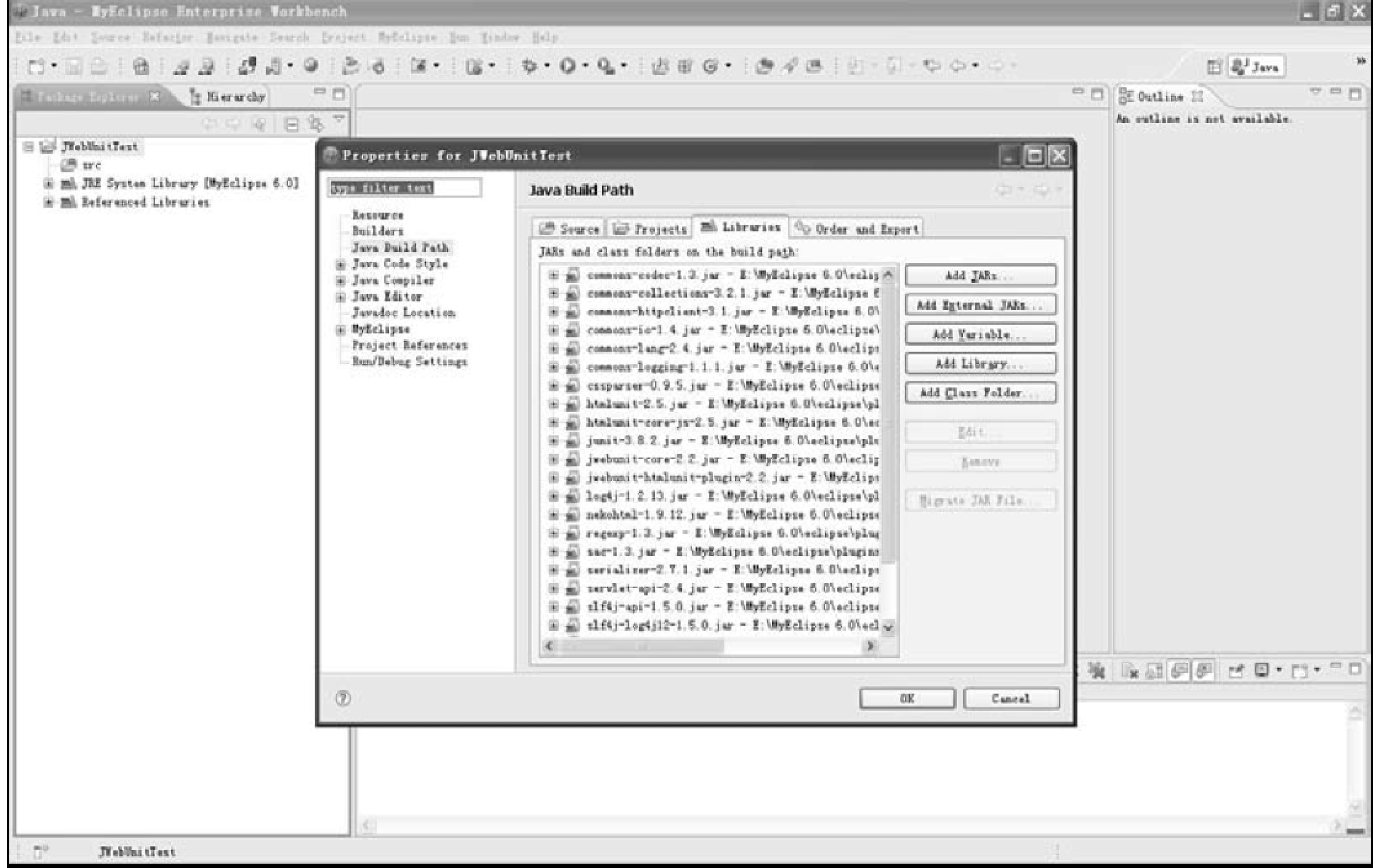


图 8-1　(续)

8.3.2　JWebUnit 应用方法

1．快速应用

JWebUnit 大致有两种方式来建立测试用例：继承模式和委托模式。下面这种是使用继

承模式继承 JWebUnit 提供的 WebTestCase 类，该类继承于 junit.framework.TestCase 类，代码示例如下：

```
import net.sourceforge.jwebunit.WebTestCase;
public class ExampleWebTestCase extends WebTestCase {
  public ExampleWebTestCase(String name) {
    super(name);
  }
}
```

委托模式的实现比上面的实现方式麻烦一点，代码示例如下：

```
import junit.framework.TestCase;
import net.sourceforge.jwebunit.WebTester;
public class ExampleWebTestCase extends TestCase {
  private WebTester tester;
  public ExampleWebTestCase(String name) {
    super(name);
    tester = new WebTester();
  }
}
```

用户测试用例代码继承了 junit.framework.TestCase 类，并在该类的全局变量里声明了一个 WebTester 类，该类封装了一些基本的 Web 操作动作，它属于 JWebUnit。在 ExampleWebTestCase 构造函数中将 WebTester 实例化。委托模式有个好处，代码表现上很简洁，测试动作和被测试业务代码将被表现得异常清晰，有利于后期的测试开发迭代。

2. HttpUnit 和 JWebUnit 测试方法对比

对 Web 应用程序自动进行测试意味着跳过 Web 浏览器，通过程序来处理 Web 站点。首先看看 HttpUnit(JWebUnit 的构建块之一)是如何简化这项工作的。HttpUnit 可以模拟帧、JavaScript、页面重定向 Cookie 等。在将 HttpUnit 用于 JUnit 时，它可以迅速地对 Web 站点的功能进行验证。

下面的代码显示了一个用 HttpUnit 编写的测试用例，它试图单击 HttpUnit 主页上的 Cookbook 链接。

```
public class HttpUnitTest {
    public static void main(String[] args) {
        try {
            WebConversation wc = new WebConversation();
            WebRequest request =
                new GetMethodWebRequest("http://httpunit.sourceforge.net/index.html");
            wc.setProxyServer( "your.proxy.com", 80 );
```

```
7              WebResponse response = wc.getResponse(request);
8              WebLink httpunitLink =
                   response.getFirstMatchingLink(WebLink.MATCH_CONTAINED_TEXT,"Cookbook");
9              response = httpunitLink.click();
10              System.out.println("Test Successful !!");
11           } catch (Exception e) {
12           System.err.println("Exception: " + e);
13        }
14  }
```

这段代码的第 6 行用 your.proxy.com 连接 Internet。如果存在直接 Internet 连接，那么可以把这条语句注释掉。第 8 行的语句用于在页面中搜索包含文本“Cookbook”的 Web 链接。第 9 行的语句用于单击这个链接。如果找到链接，用户将会看到“Test Successful!”这条消息。

下面的代码是用 JWebUnit 改写后的上述 HttpUnit 的测试代码，看上去更简单了。

```
1 public class JWebUnitTest extends WebTestCase{
2     public static void main(String[] args){
3         junit.textui.TestRunner.run(new TestSuite(JWebUnitTest.class));
4     }
5     public void setUp(){
6         getTestContext().setBaseUrl("http://httpunit.sourceforge.net");
7         getTestContext().setProxyName("webproxy.watson.ibm.com");
8         getTestContext().setProxyPort(8080);
9     }
10    public void testSearch(){
11    beginAt("/index.html");
12    clickLinkWithText("Cookbook");
13    }
14 }
```

如果没注意特定于 JUnit 的代码，那么可以看到，测试用例现在变得相当整洁、简练。需要查看的重要的行是第 6 行、第 11 行和第 12 行。

在第 6 行，基本 URL 被设置到 HttpUnit 的主页中。第 11 行用相对路径 /index.html 连接站点。第 12 行用于单击页面上具有文本 Cookbook 的链接。如果链接有效，那么 JUnit 会报告成功；否则，JUnit 会报告异常。

3. JWebUnit API

每个 JWebUnit 测试的核心都是 net.sourceforge.jwebunit.WebTestCase 类，它代表测试用例。每个测试用例都必须是从这个类扩展而来的(net.sourceforge. jwebunit.WebTestCase 类本身则是从 junit.framework.TestCase 类扩展而来的，它在 JUnit 中代表测试用例)。

WebTestCase 类的重要方法参见表 8-1。

表 8-1　net.sourceforge.jwebunit.WebTestCase 类的重要方法

方　　法	说　　明
public TestContext getTestContext()	得到测试用例的上下文。可以用它访问诸如地区、基本 URL 和 Cookie 之类的项目
public void beginAt(String relativeURL)	在相对于基本 URL 的 URL 处开始对话
public void setWorkingForm(String nameOrId)	与指定的表单开始交互。如果当前页面只有一个表单，就不需要调用这个方法
protected void submit()	提交表单——等同于单击表单的“提交”按钮
public void gotoFrame(String frameName)	激活命名帧

另一个重要的类是 net.sourceforge.jwebunit.TestContext，它用于为测试创建上下文。可以用这个类来处理诸如 Cookie、会话和授权之类的信息。表 8-2 显示了这个类的一些重要方法。

表 8-2　net.sourceforge.jwebunit.TestContext 类的重要方法

方　　法	说　　明
public void addCookie(String name, String value)	向测试上下文中添加 Cookie。在 HttpUnitDialog 开始时，添加的 Cookie 被设置到 WebConversation 上
public void setResourceBundleName(String name)	为测试上下文设置一个使用的资源绑定。用于按照 WebTester 中的键查找期望的值
public void setProxyName(String proxyName)	为测试上下文设置代理服务器名称
public void setBaseUrl(String url)	为测试上下文设置基本 URL

8.3.3　JWebUnit 测试应用举例

现在通过例子来介绍 JWebUnit API 的实际应用。我们考察的应用程序是一个测试用例，用于打开一个 Google 搜索页面并搜索文本“HttpUnit”。应用程序需要测试以下场景。

(1) 打开 Google 主页 http://www.google.com。

(2) 确定该页包含一个名为 q 的表单元素(在 Google 的主页上，名为 q 的文本框用于接收用户的查询输入条件)。应用程序用这个元素输入搜索参数。

(3) 在搜索文本框中输入字符串“HttpUnit Home”，并提交表单。

(4) 获得结果页，并确定该页面包含的链接中包含文本“HttpUnit Home”。

(5) 单击包含文本 HttpUnit Home 的链接。

现在测试场景已经就绪，可以编写 Java 应用程序，用 JWebUnit 实现这些需求了。

第一步是声明一个从 WebTestCase 扩展而来的类，如(1)所示。

(1) 声明测试用例类：

```
public class GoogleTest extends WebTestCase {
    static String searchLink = "";
}
```

正如在前面提到过的，JWebUnit 要求每个测试用例都是从 WebTestCase 中扩展而来的。searchLink 保存传入的搜索参数。这个值以命令行参数的形式传递给测试用例。

下一步是声明入口点——main()方法，如(2)所示。

(2) 声明 main()方法：

```
public static void main(String[] args) {
    searchLink = args[0];
    junit.textui.TestRunner.run(new TestSuite(GoogleTest.class));
}
```

main()方法调用 junit.textui.TestRunner.run()来执行测试用例。因为需要运行 GoogleTest 测试用例，所以作为参数传递给 run()方法的测试套件采用 GoogleTest.class 作为参数。

接下来，浏览用例调用 setUp()方法来设置基本 URL 和代理，如(3)所示。

(3) 设置基本 URL 和代理：

```
public void setUp() {
    getTestContext().setBaseUrl("http://www.google.com");
    getTestContext().setProxyName("proxy.host.com");
    getTestContext().setProxyPort(80);
}
```

上面把基本 URL 设置为 http://www.google.com，这意味着测试用例的启动是相对于这个 URL 的。接下来的两条语句设置连接到 Internet 的代理主机和代理端口，如果是直接连接到 Internet，那么可以忽略代理设置语句。

现在开始浏览站点并输入搜索参数。

(4) 显示了访问 Web 页面，然后测试所有场景的代码。

(5) 测试所有场景：

```
public void testSearch() {
    beginAt("/");
    assertFormElementPresent("q");
    setFormElement("q", "HttpUnit");
    submit("btnG");
    assertLinkPresentWithText(searchLink);
    clickLinkWithText(searchLink);
}
```

上述代码连接到基本 URL，并相对于“/”开始浏览。然后它断定页面中包含一个名为 q 的表单元素——q 是 Google 主页上查询输入文本框的名称。下一条语句用值 HttpUnit

来设置名为 q 的文本框。再下一条语句提交表单上名为 btnG 的“提交”按钮(在 Google 主页上，名为 btnG 的按钮是标签为“Google Search”的按钮)。表单是在这个对话中提交的，下一页列出搜索结果。在结果页面上，代码首先检查是否有一个链接的文本是“HttpUnit Home”。如果该链接不存在，那么测试就以 AssertionFailedError 失败。如果该链接存在，则测试执行的下一个操作是单击链接。

在测试环境中编写上述的完整 Java 测试程序：

```
package com.jweb.test;
import junit.framework.TestSuite;
import junit.textui.TestRunner;
import net.sourceforge.jwebunit.TestContext;
import net.sourceforge.jwebunit.WebTestCase;
public class GoogleTest extends WebTestCase
{
    static String searchLink = "";
    public static void main(String[] args) {
        searchLink = args[0];
        TestRunner.run(new TestSuite(GoogleTest.class));
    }
    public void setUp() {
        getTestContext().setBaseUrl("http://www.google.com");
        getTestContext().setProxyName("proxy.host.com");
        getTestContext().setProxyPort(80);
    }
    public void testSearch() {
        beginAt("/");
        assertFormElementPresent("q");
        setFormElement("q", "HttpUnit");
        submit("btnG");
        assertLinkPresentWithText(searchLink);
        clickLinkWithText(searchLink);
    }
}
```

在控制台执行 java com.jweb.test.GoogleTest "HttpUnit Home"命令，运行程序。

在执行了测试用例之后，会在命令行输出一个测试用例报告。如果测试失败，报告将看起来如(6)中所示。

(6) 带有断言失败的输出：

```
C:\temp>java com.jweb.test.GoogleTest "HttpUnit Hwee"
.F
Time: 5.338
```

```
There was 1 failure:
1) testSearch(com.jweb.test.GoogleTest)junit.framework.AssertionFailedError: Link
   with text [HttpUnit Hwee] not found in response.
        at net.sourceforge.jwebunit.WebTester.assertLinkPresentWithText(WebTester.java:618)
        at net.sourceforge.jwebunit.WebTestCase.assertLinkPresentWithText(WebTestCase.java:244)
        at com.jweb.test.GoogleTest.testSearch(GoogleTest.java:36)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at com.jweb.test.GoogleTest.main(GoogleTest.java:19)
FAILURES!!!
Tests run: 1,   Failures: 1,   Errors: 0
```

正如在(6)中可以看到的，可以以 HttpUnit Hwee 作为参数来执行测试用例。这个测试用例遇到断言的地方会失败，因为结果页面中不包含带有这个文本的链接。由此也就产生了 junit.framework.AssertionFailedError。

在(7)中执行时以 HttpUnit Home 作为参数。测试用例找到了一个带有这个文本的链接，所以测试通过了。

(7) 成功测试的输出：

```
C:\temp>java com.jweb.test.GoogleTest "HttpUnit Home"
.
Time: 6.991
OK (1 test)
```

8.3.4　JWebUnit 应用小结

前面通过讨论 JWebUnit 框架的一些突出特性和最重要的类，介绍了如何用它创建简洁的测试用例，使读者能够对 JWebUnit 框架有一个认识。JWebUnit 还有更多特性可以用在测试用例中。它支持测试 Web 页面中的链接行数，可以对字符串、表或者带有指定标签的表单输入元素是否存在于页面上进行断言。此外，JWebUnit 还可以处理 Cookie(例如断言存在某个 Cookie、删除 Cookie 等)。测试可以对某个文本之后出现的特定文本的链接进行单击。如果想为 Web 应用程序构建快而有效的测试用例，JWebUnit 可能是最好的工具。

8.4　Web 页面测试工具之三——Selenium

Selenium 是一个用于 Web 应用程序测试的工具，通过模拟用户对 Web 页面的各种操作，可以精确重现软件测试人员编写的 Test Cases 步骤。Selenium 测试直接运行在浏览器

中，就像真正的用户在操作一样。支持的浏览器包括 IE、Mozilla Firefox、Mozilla Suite 等。其他测试工具都不能覆盖如此多的平台。

使用 Selenium 和在浏览器中运行测试还有很多其他好处。下面是主要的两大好处。

(1) 通过编写模仿用户操作的 Selenium 测试脚本，可以从终端用户的角度来测试应用程序。

(2) 通过在不同浏览器中运行测试，更容易发现浏览器的不兼容性。

Selenium 的核心，也称 browser bot，是用 JavaScript 编写的。这使得测试脚本可以在受支持的浏览器中运行。browser bot 负责执行从测试脚本接收到的命令，测试脚本要么是用 HTML 的表布局编写的，要么是使用一种受支持的编程语言编写的。

Selenium 的主要功能包括：测试与浏览器的兼容性——测试应用程序看是否能够很好地工作在不同浏览器和操作系统之上。测试系统功能——检验软件功能和用户需求。支持自动录制操作和自动生成.NET、Java、Perl 等不同语言的测试脚本。Selenium 是 ThoughtWorks 专门为 Web 应用程序编写的一个集成测试、系统测试及验收测试的测试工具。

Selenium 包含三个工具：Selenium-IDE，Selenium-RC 以及 Selenium-Core。其中，Selenium-Core 是驱动 Selenium 工作的核心部分，作为一个用 JavaScript 编写的测试引擎，它可以操作 Web 页面上的各种元素，诸如：单击按钮、输入文本框，以及断言 Web 页面上存在某些文本与 Web 元素等，支持 Windows 平台上的 IE、Firefox、Chrome 等浏览器以及 Windows 和 Linux 平台。

Selenium-IDE 是一个 Firefox 插件，能够录制回放用户在 Firefox 中的行为，并把所记录的 Selenese(Selenium Commands)转为 Java/C#/Python/Ruby 等语言，在 Selenium-RC 中修改重用。对于较为复杂的 Test Cases，Selenium-IDE 的功能有限，往往用它录制大致的步骤，再转化为测试人员熟悉的编程语言，在此基础上完善，形成更为强大且灵活的 Selenium-RC Test Cases。

Selenium-RC（Selenium Remote Control）在 Web 浏览器与需要测试的 Web 应用间架设代理服务器（Selenium Server），使得 JavaScript 引擎与被测 Web 应用同源，绕开同源策略的限制（Same Origin Policy），进而取得对 Web 页面进行各种操作的权限。

8.4.1 Selenium 环境建立

这里选用的是最新版本的 Selenium 组件，Selenium 2.0.0 版本，与其匹配的火狐浏览器为 Firefox 19.0 或者 Firefox 20.0。

1. 安装 Firefox

下载 Firefox 火狐浏览器 20.0，下载地址为http://firefox.com.cn/download/，下载后安装在系统默认路径 C:\Program Files\Mozilla Firefox。

2. 安装 Selenium

下载 Selenium IDE，下载地址为http://docs.seleniumhq.org/download/，下载后将 xpi 文件复制到 Firefox 安装目录的 extensions 目录，路径为 C:\Program Files\Mozilla Firefox\extensions，然后重启 Firefox，在菜单“工具”中显示 Selenium IDE，单击该菜单项，弹出 Selenium IDE 2.0.0 窗口，如图 8-2 所示，说明安装成功。

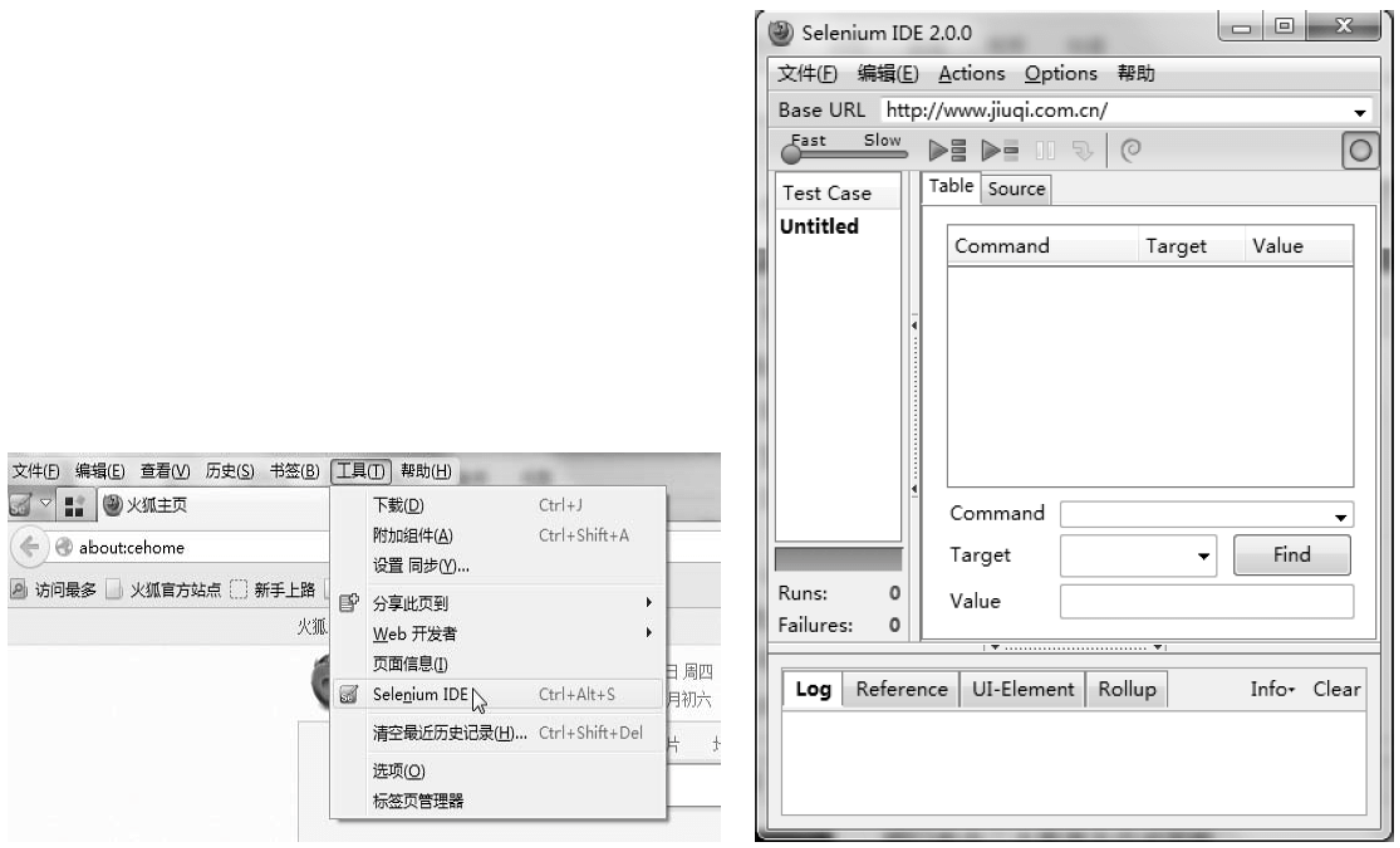


图 8-2　Selenium 安装成功

3. 启动测试服务

启动久其软件官方网站服务，在 Firefox 浏览器中输入“http://www.jiuqi.com.cn/”，显示如图 8-3 所示，证明服务启动成功。



图 8-3　服务启动成功

8.4.2　应用流程

1. 新建测试集

测试集是一组测试用例，是 Selenium IDE 可以播放的最大单元，新建测试集非常简单，在 Selenium IDE 的菜单中单击 New Test Suite 命令即可，如图 8-4 所示。

2. 新建测试用例

测试用例是 Selenium 管理的最小单元，一个测试集可以有多个测试用例，测试用例可以单个播放，也可以一个套件播放，新建测试用例也非常简单，只需要单击 Selenium IDE 菜单中的 New Test Case 命令即可，如图 8-5 所示。

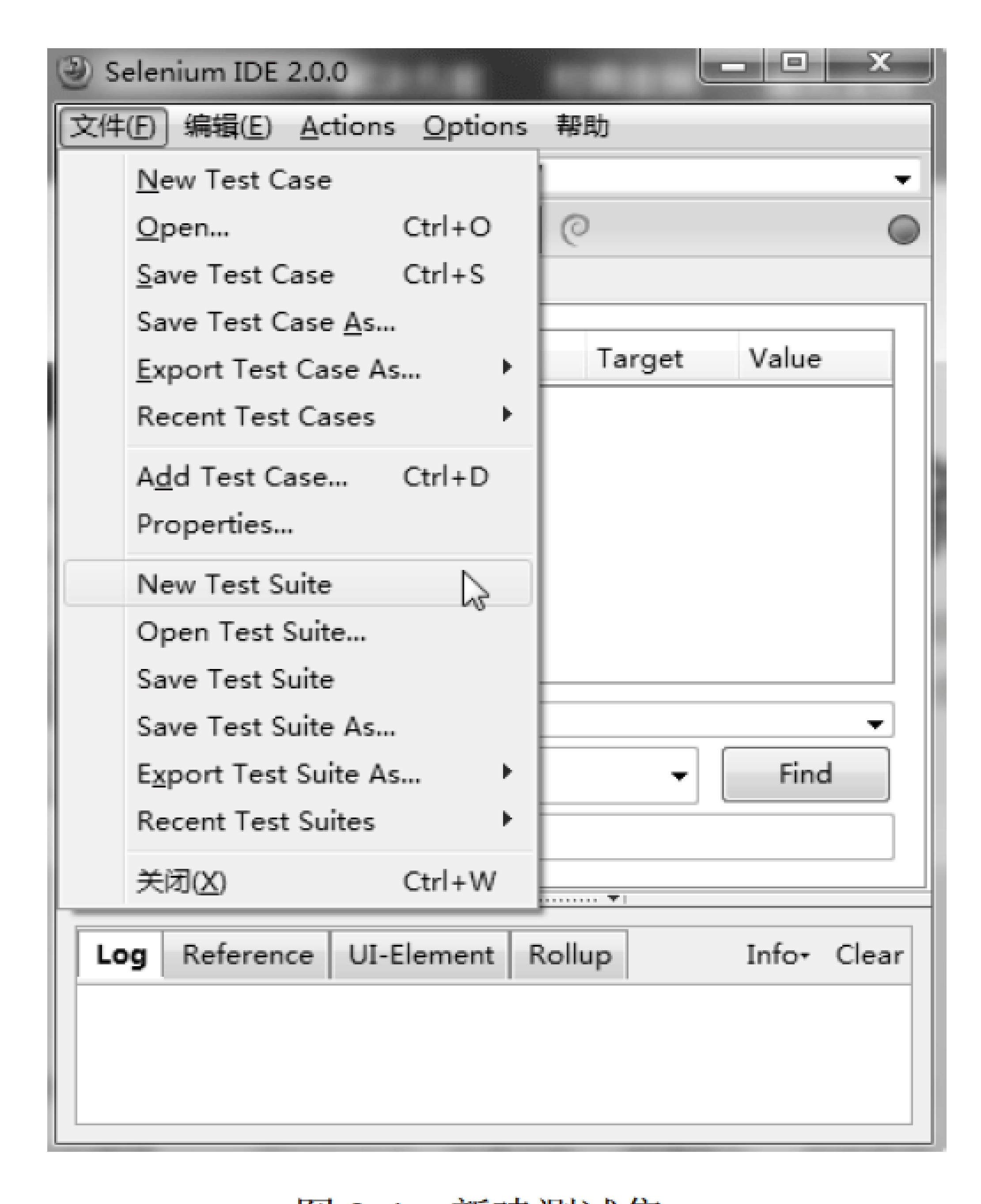




图 8-4　新建测试集

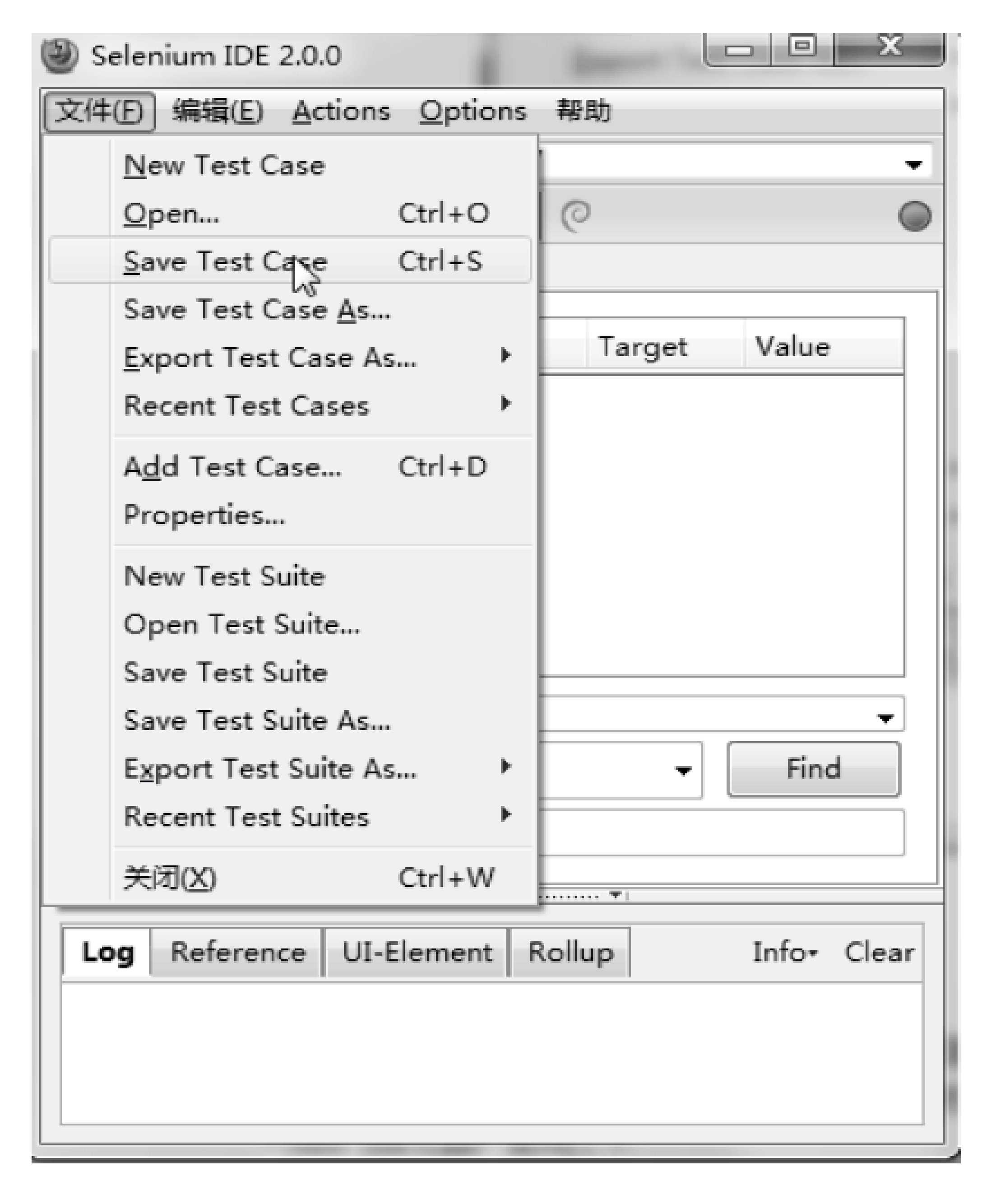


图 8-5　新建测试用例

3. 录制测试脚本

创建了测试用例就可以录制该用例脚本，单击 IDE 界面中的红色按钮，启动脚本录制，如图 8-6 所示。



图 8-6　启动脚本录制

然后在系统界面中执行相应的操作，直到该用例完成，再次单击红色按钮，结束脚本

的录制，IDE 界面中将显示该用例对应的脚本内容，如图中的 Table 选项卡，其中包含 Command、Target、Value 等信息，如图 8-7 所示。

4. 保存测试用例

测试用例脚本录制完毕后，可以将脚本进行保存，只需要单击 IDE 菜单中的 Save Test Case 命令，然后提供以下测试用例的名称，比如“测试用例 01”，单击“确定”即可，指定目录中会多一个 html 后缀的文件。保存后 IDE 界面左侧显示了保存后的测试用例名称，选中后右侧界面显示了对应的脚本信息，如图 8-8 所示。

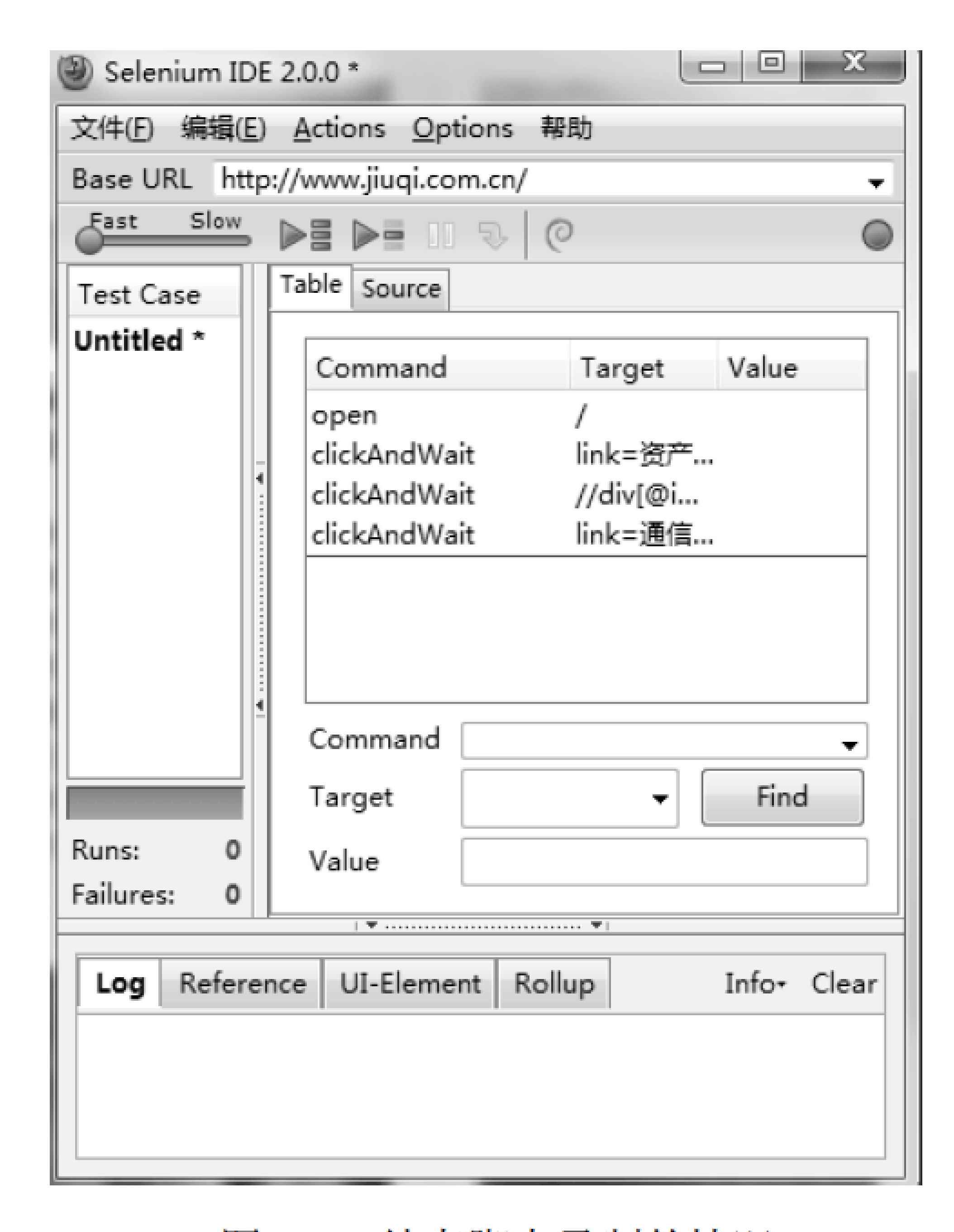


图 8-7　结束脚本录制的情况

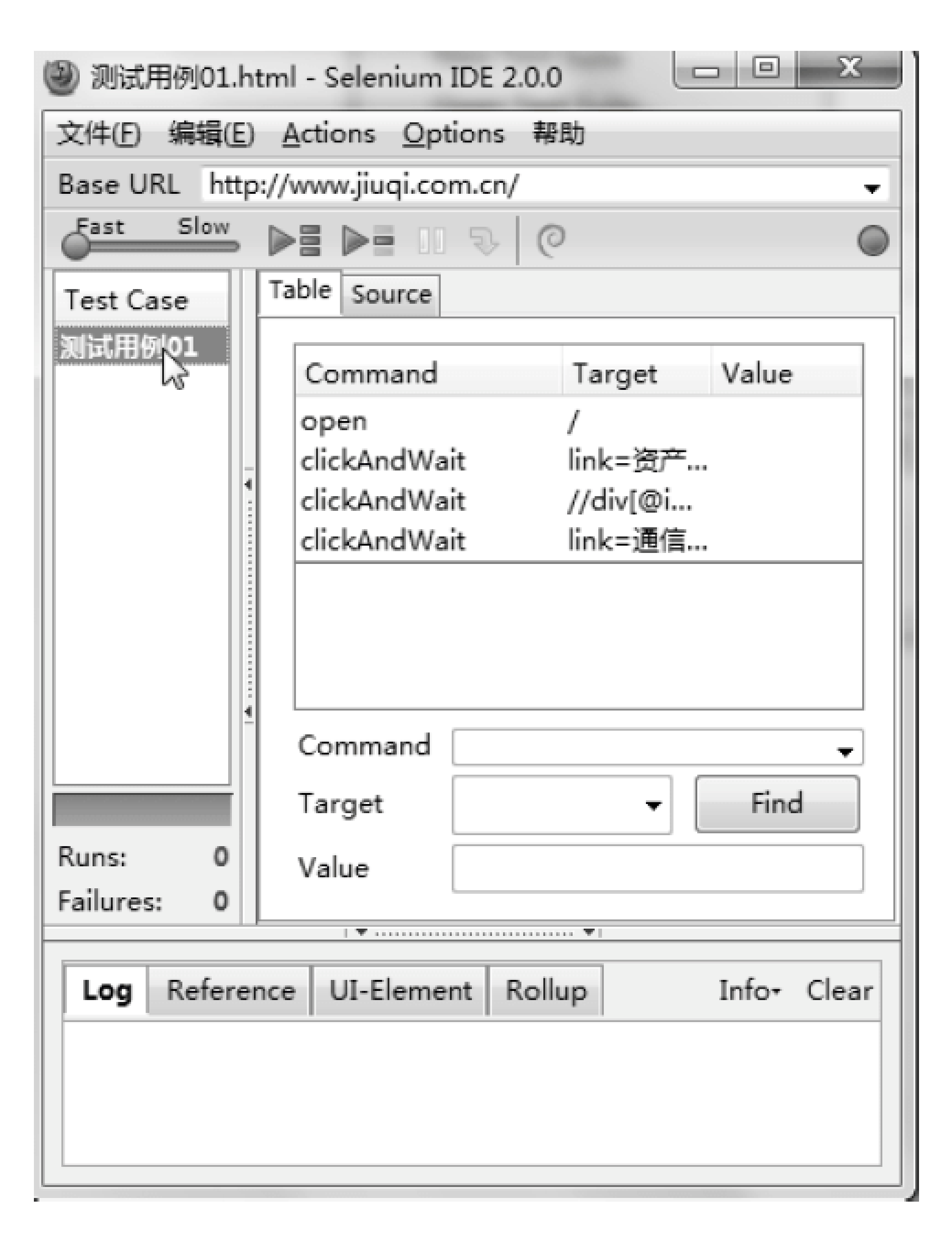


图 8-8　测试用例的保存情况

5. 保存测试集

保存了多个测试用例后，可以将这些测试用例保存到一个测试集中，只需要单击 IDE 菜单中的 Save Test Suite 命令，提供以下测试集的名称，比如“测试集 01”，单击“确定”即可，指定目录中会多出一个没有后缀名的文件。

6. 打开测试集

保存后的测试集可以被 IDE 再次打开，单击 IDE 菜单中的 Open Test Suite 命令，选择测试集所在的目录，选中后单击“确定”即可。打开后 IDE 界面中将显示该测试集的用例信息，如图 8-9 所示。

7. 回放测试用例

录制的测试用例脚本可用来修改、补充和回放。在 IDE 中准备好基本的脚本后，确定输入内容正确无误，也做好了验证设定，可以回放当前脚本，最终 IDE 会给出提示通过情

况和不通过情况。

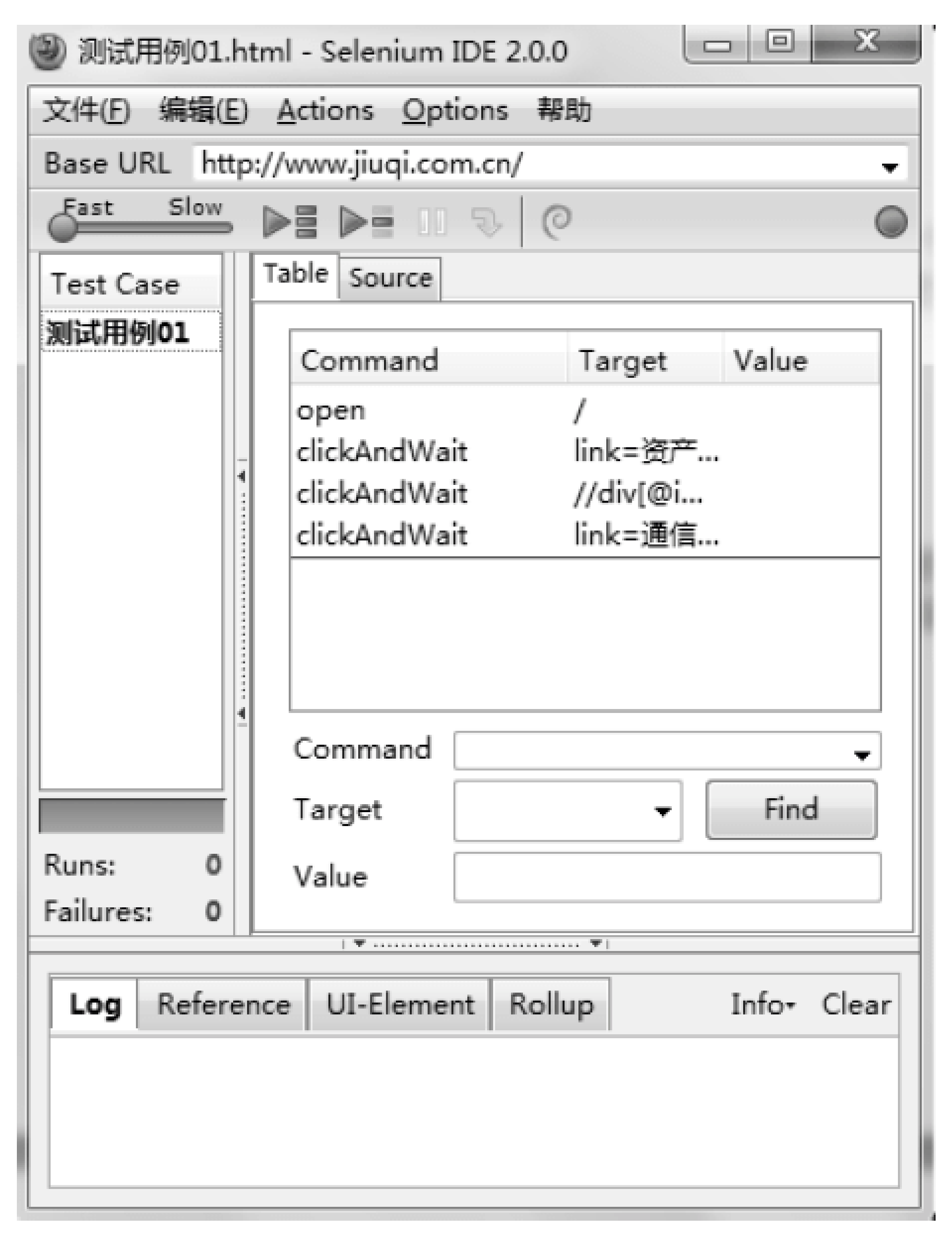


图 8-9　测试集的打开

在 IDE 的录制完成之后回放，经常会遇到回放失败的过程，所以需要对 IDE 的脚本录制进行修改，在录制回放中的经验主要涉及以下几点。

(1) 定位不准，如果 xpath 定位不准则自己根据页面输入 xpath，也可以通过 ViewPath 和 FireBug（都是火狐的插件）来定位 XPath 脚本，即 Target。

(2) click 是最常用的单击事件，如果在回放的过程中不执行 click 事件可以试试其他的事件比如 MouseDown 等。

(3) 有时打开了多个页面，IDE 无法定位哪个页面为当前的脚本执行页面，需要使用 SelectWindow 来定位。

IDE 提供了两种回放方式，即回放单个测试用例和回放当前测试集中的所有测试用例，并且可以设置回放的速度，回放完毕后，界面将显示用例回放的结果，包括回放用例的个数、失败的个数等，如图 8-10～图 8-13 所示。



图 8-10　播放单个测试用例

图 8-11　播放测试集所有用例

图 8-12　设置测试用例的播放速度

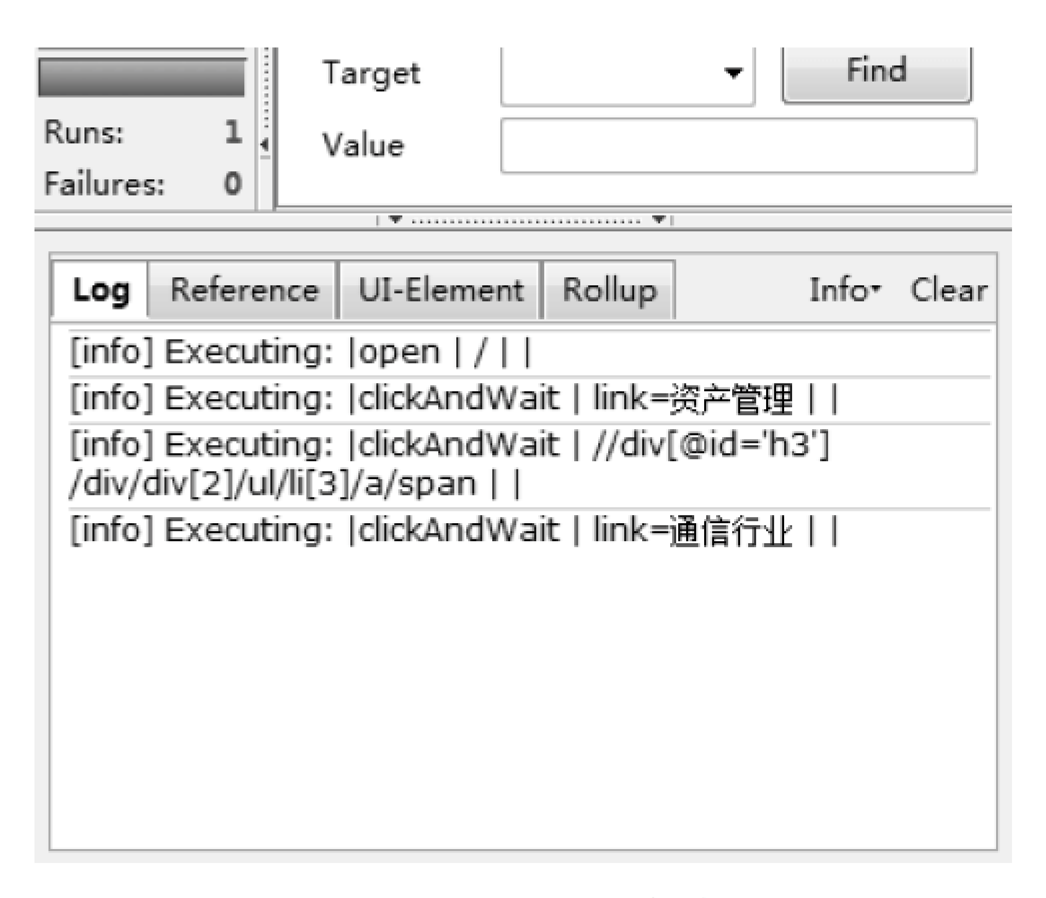


图 8-13　显示回放结果

8. 基于 IDE 调试脚本

在 IDE 上调试脚本如图 8-14 所示。

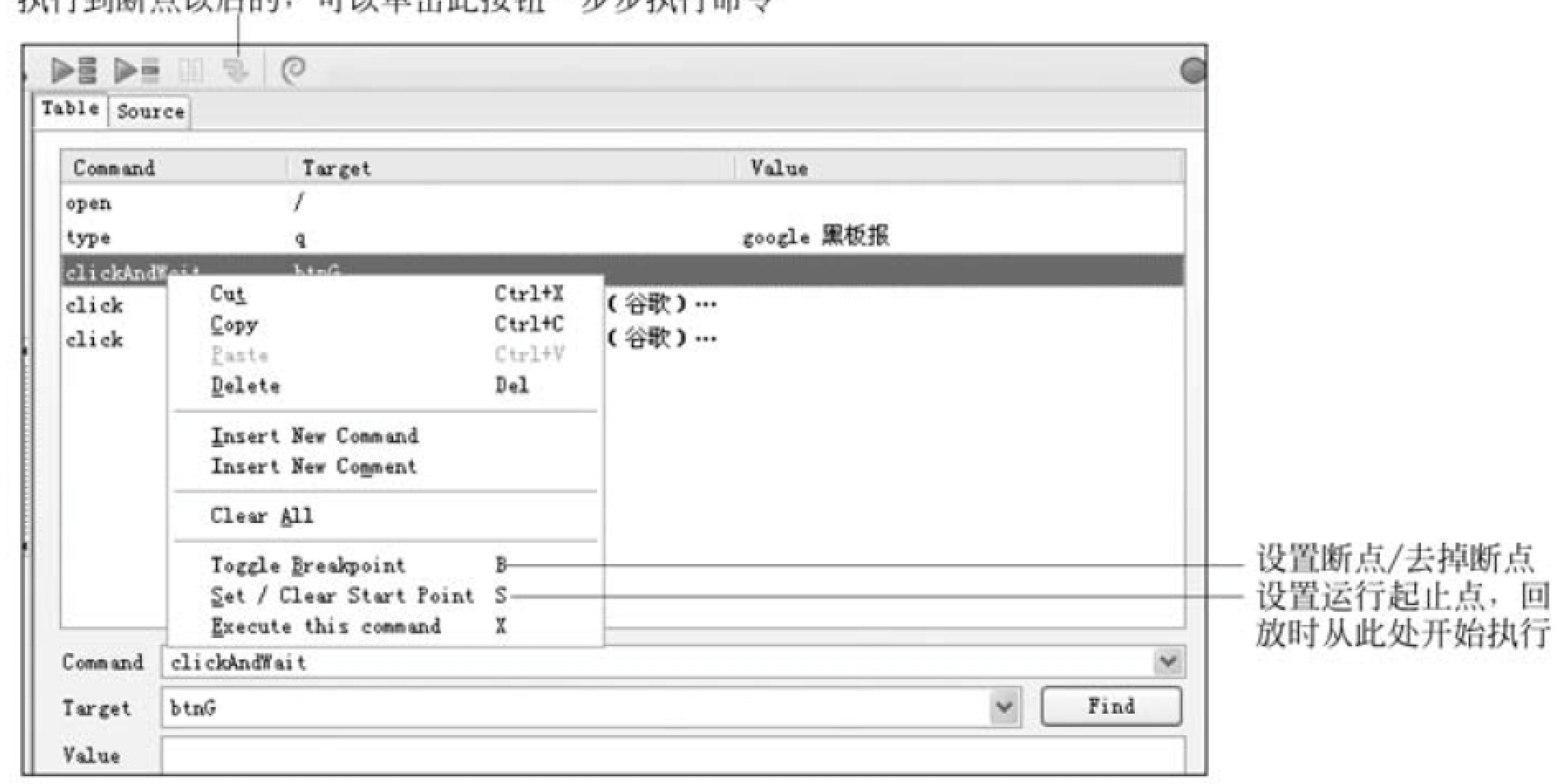


图 8-14　脚本调试

9. 把录制的内容转化成 Java 脚本

将录制的内容转化成 Java 脚本的方法如图 8-15 所示。

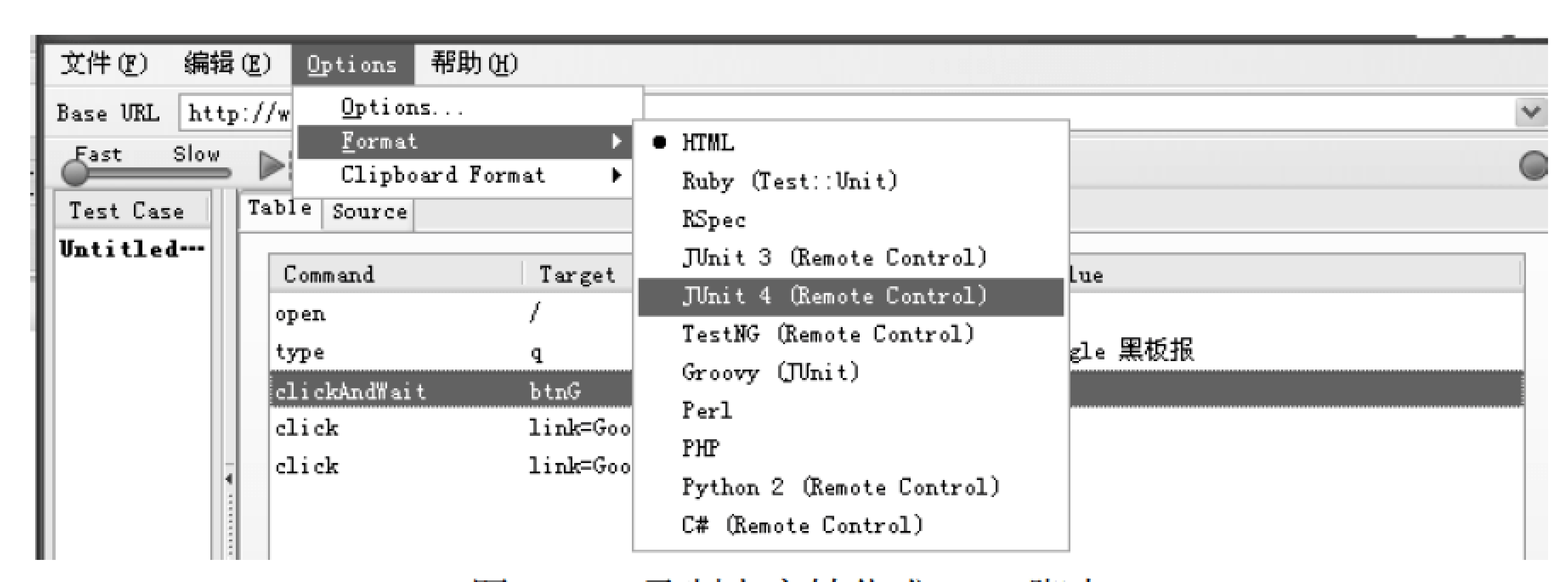


图 8-15　录制内容转化成 Java 脚本

8.4.3　应用举例

1. 系统需求

上市公司久其软件官方网站中的“关于久其”功能，包括“公司简介”、“发展历程”、“企业文化”、“久其期刊”、“资质荣誉”、“新闻中心”、“营销网络”、“合作伙伴”，其中“合作伙伴”是弹出式窗口，其他功能都是在本窗口中显示。

2. 脚本录制

(1) 用 Firefox 访问久其软件官网，进入“关于久其”功能。

(2) 单击 Firefox 菜单栏的“工具”|Selenium IDE 命令，打开 Selenium IDE 窗口，单击红色按钮，开始测试脚本录制。

(3) 回到官网界面，依次单击“公司简介”、“发展历程”、“企业文化”、“久其期刊”、“资质荣誉”、“新闻中心”、“营销网络”、“合作伙伴”。

(4) 再次单击红色按钮，结束测试脚本的录制。保存测试用例，用例名为“关于久其”；并保存测试集，测试集命名为“久其软件官网_关于久其”。

3. 脚本回放

打开测试集“久其软件官网_关于久其”，选中“关于久其”测试用例，单击“回放测试用例”按钮，执行测试脚本回放，回放完成后，界面输出以下内容。

(1) 久其官网操作表，如图 8-16 所示。

Command	Target
open	/aboutus/index.jhtml
clickAndWait	link=公司简介
clickAndWait	link=发展历程
clickAndWait	link=企业文化
clickAndWait	link=久其期刊
clickAndWait	link=资质荣誉
clickAndWait	link=新闻中心
clickAndWait	link=营销网络
clickAndWait	link=合作伙伴

图 8-16　界面操作情况

(2) 操作生成的脚本程序：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://www.jiuqi.com.cn/" />
```

```
<title>New Test</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">New Test</td></tr>
</thead><tbody>
<tr>
	<td>open</td>
	<td>/aboutus/index.jhtml</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=公司简介</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=发展历程</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=企业文化</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=久其期刊</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=资质荣誉</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=新闻中心</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>link=营销网络</td>
```

```
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>link=合作伙伴</td>
        <td></td>
</tr>
</tbody></table>
</body>
</html>
```

(3) 回放生成的日志如图 8-17 所示。

(4) 回放的结果如图 8-18 所示。

```
[info] Executing: |open | /aboutus/index.jhtml | |
[info] Executing: |clickAndWait | link=公司简介 | |
[info] Executing: |clickAndWait | link=发展历程 | |
[info] Executing: |clickAndWait | link=企业文化 | |
[info] Executing: |clickAndWait | link=久其期刊 | |
[info] Executing: |clickAndWait | link=资质荣誉 | |
[info] Executing: |clickAndWait | link=新闻中心 | |
[info] Executing: |clickAndWait | link=营销网络 | |
[info] Executing: |clickAndWait | link=合作伙伴 | |
```

图 8-17　操作日志



图 8-18　回放结果

4. 总结

Selenium 是软件工程师、设计人员和测试人员的工具箱中又一个有用且重要的工具。通过将该工具与持续集成工具相结合，团队就可以将验收测试自动化，并构建更好的软件，因为他们可以更容易、更早、更频繁地发现 Bug。Selenium 的另一个优点是可以节省时间，使开发人员和测试人员不必将时间花在本可以（也应该）自动化的手工任务上，从而让团队将精力放在更有价值的活动上。

实 验 习 题

1. 应用 HtmlUnit 对 Web 应用进行测试，并给出具体的测试过程，同时与 HttpUnit 进行比较。

2. 在 Eclipse 环境下建立 swtbot 的 Web 应用测试环境，并对具体的 Web 应用进行测试，详细描述测试过程。

第9章　Gtk+用户界面测试

前面章节主要介绍的是有关用 Java 开发的界面测试技术、方法和工具，并且无论是商用的还是开源的，关于 GUI 的自动化测试工具基本上都是针对 Windows 程序的，很少有针对 Linux 图形用户界面程序的。本章将以 Linux 主流界面工具 Gtk+为例，介绍不知名但较实用且能够反映界面测试的一般方法和开源工具。

Gtk(GIMP Toolkit)+虽然是使用 C 语言开发的，但是其设计者却使用了面向对象技术。它是一套跨多种平台的图形工具包，并且是按 LGPL 许可协议发布的。虽然最初是为 GIMP 编写的，但目前已发展为一个功能强大、设计灵活的通用图形库。特别是被 GNOME 选中后，使得 Gtk+广为流传，成为 Linux 下开发图形界面应用程序的主流开发工具之一。当然 Gtk+并不要求必须是在 Linux 上使用，事实上，目前 Gtk+已经有了成功的 Windows 版本。

Gtk+虽然是用 C 语言编写的，但是用户也可以使用自己熟悉的语言来使用它，因为 Gtk+已经被绑定到几乎所有流行的语言上，如 C++、Guile、Perl、Python、Java、TOM、Ada95、Objective C、Free Pascal、Eiffel 等。

目前有很多系统环境或系统工具，以及应用开发所用到的 GUI 开发工具都是 Gtk+。在 Gtk+界面开发过程中，人们常用到 Glade。Glade 是一个界面设计工具，它还包含一种描述 GUI 界面的 XML，它和 libglade 一起工作就可以直接使用 Gtk+和 GNOME 控件了。

Gtk+提供不同的显示引擎，以使最终用户可以定制外观和感觉。目前已经有一些可以模仿其他流行平台或工具箱(例如 Windows、Motif、Qt 或 NEXTSTEP)的引擎。

2002 年后，Gtk+ 2.0 版正式发布。Gtk+ 2 是 Gtk+的后继版本，其新特性包括使用 Pango 改进的文本渲染、新主题引擎、使用 ATK 改进的可达性、完全转换到使用 UTF-8 的 Unicode 和更灵活的 API。但是它和 Gtk+不完全兼容，因此必须由程序员做移植工作。但由于 Gtk+更快、相对更简单和更加适合嵌入式应用，所以它还被继续使用着。

从 Gtk+ 2 的 2.8 版起，它依靠库 Cairo 来完成渲染，而这引入了矢量图形的支持。Gtk+目前已升级到了 Gtk+3.x。

Gtk+作为界面设计工具具有很多优势：①不仅现代，而且得到了积极的开发与维护；②提供了广泛的选项，用于把工作扩展到尽可能多的人，其中包括一个针对国际化、本地化和可访问性的完善的框架；③简单易用，对开发人员和用户来说都是如此；④设计良好、灵活而可扩展；⑤是自由软件，有一个自由的开放源码许可；⑥从用户和开发人员的角度看是可移植的。

9.1　Gtk+用户界面概述

1. Gtk+组成

Gtk+由 4 部分组成：GDK、gdk-pixbuf、GLib 和 Pango。

Gtk+使用 Glib 库和 GDK(GIMP Drawing Kit，GIMP 绘图工具包)系列的开发库，Glib 库定义了数据类型，提供了错误处理和内存管理方面的函数；而 GDK 则是本地图形化 API 与 Gtk+ 之间的一个过渡层，它需要依赖具体的计算机平台。因此，向其他计算机平台上移植 Gtk+时，只需要重新编写 GDK 即可。

1) GDK 和 gdk-pixbuf

Gtk+称为 GIMP 工具包是因为最初编写它是用来开发 GIMP (GNU 图像处理程序)的，但是它现在已经被用于很多软件项目了，包括 GNOME (GNU 网络对象模型环境)。Gtk+是在 GDK (GIMP Drawing Kit)和 gdk-pixbuf 的基础上建立起来的，GDK 基本上是对访问窗口的底层函数(在 X 窗口系统中是 Xlib)的一层封装，gdk-pixbuf 是一个用于客户端图像处理的库。Gtk+本质上是一个面向对象的应用程序接口(API)，尽管是完全用 C 写成的，但它却是基于类和回调函数(指向函数的指针)思想实现的。

2) Glib

最初 Gtk+仅包括一些和图形无关的常规功能，如链表和二叉树等数据结构。这些基本功能和对象系统 GObject 已经合并到独立的库 Glib，它被程序员专门用于开发不需要图形界面的代码。Glib 包含一些标准函数的替代函数，以及一些处理链表等数据结构的函数等。这些替代函数被用来增强 Gtk 的可移植性，因为它们所实现的一些函数在其他 UNIX 系统上未实现或不符合标准，比如 g_strerror()。还有一些是对 libc 的对应函数的增强，比如 g_malloc()就具有增强的调试功能。在 2.0 版中，Glib 又加入了以下新内容：构成 Gtk 类层次基础的类型系统(type system)，在 Gtk 中广泛使用的信号系统，对各种不同平台的线程 API 进行抽象而得到的一个线程 API，以及一个用于加载模块的工具。

3) Pango

作为最后一个组件，Gtk 使用 Pango 库来处理国际化文字输出。

还有许多其他语言对 Gtk 进行了绑定，如 C++、Perl、Python、TOM、Ada95、Objective C、Free Pascal、Eiffel、Java、C#等。如果想使用 Gtk 其他语言的绑定，请先查看该绑定的文档，有时这些文档会介绍一些重要的概念。还有一些跨平台的 API(如 wxWindows 和 V)，它们把 Gtk 作为一个支持的平台。

2. Gtk+安装

一般来说，Linux 下均安装了 Gtk+工具库(无论是 Gtk+ v1.2 还是 Gtk+ v2.0)，可以直

接在上面开发程序。如果当前系统没有 Gtk+工具库，可以到 www.gtk.org 下载源码安装。

对于 Gtk+ v1.2.10 来说，需要先安装支持其工作的 glib v1.2.10。例如，先将 glib v1.2.10.tar.gz 解压缩到 root 目录下，在 root 目录下找到其相应的解压的文件夹目录并进入，查看 readme 文档中的 Glib 安装步骤。默认情况下是执行如下命令进行系统适配、编译和安装：

```
./configure
make
make install
```

Gtk+的安装也同样需要对 gtk+ v1.2-10.tar.gz 进行解压缩、系统适配、编译和安装。

3. Gtk+应用举例

下面使用 Gtk 来编写经典的 Hello World 程序，即写一个只有一个按钮构件的程序，这是一个标准的 Gtk Hello World 程序，如图 9-1 所示。

图 9-1　Hello World 界面

```
#include <gtk/gtk.h>
/* 这是一个回调函数。data 参数在本示例中被忽略。
 * 后面有更多的回调函数示例。*/
void hello( GtkWidget *widget, gpointer   data )
{
    g_print ("Hello World\n");
}
gint delete_event( GtkWidget *widget, GdkEvent  *event, gpointer   data )
{
    /* 如果 delete_event 信号处理函数返回 FALSE，Gtk 会发出 "destroy" 信号。
     * 返回 TRUE，不希望关闭窗口。
     * 当想弹出 "您确定要退出吗?" 对话框时它会很有用。*/
    g_print ("delete event occurred\n");
    /* 改 TRUE 为 FALSE，程序会关闭。*/
    return TRUE;
}
/* 另一个回调函数 */
void destroy( GtkWidget *widget, gpointer   data )
{
    gtk_main_quit ();
}
int main( int   argc, char *argv[] )
{
    /* GtkWidget 是构件的存储类型 */
    GtkWidget *window;
    GtkWidget *button;
    /* 这个函数在所有的 Gtk 程序中都要调用。参数由命令行中解析出来并送到该程序中*/
```

```
    gtk_init (&argc, &argv);
    /* 创建一个新窗口 */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    /* 当窗口收到 delete_event 信号 (这个信号由窗口管理器发出，通常是由“关闭”
     * 选项或标题栏上的“关闭”按钮发出的)，让它调用在前面定义的 delete_event() 函数。
     * 传给回调函数的 data 参数值是 NULL，它会被回调函数忽略。*/
    g_signal_connect(G_OBJECT (window), "delete_event", G_CALLBACK (delete_event), NULL);
    /* 在这里连接 destroy 事件到一个信号处理函数。
     * 对这个窗口调用 gtk_widget_destroy()函数或在 delete_event 回调函数中返回 FALSE 值，
     * 都会触发这个事件。*/
    g_signal_connect (G_OBJECT (window), "destroy", G_CALLBACK (destroy), NULL);
    /* 设置窗口边框的宽度。*/
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    /* 创建一个标签为 "Hello World" 的新按钮。*/
    button = gtk_button_new_with_label ("Hello World");
    /* 当按钮收到 "clicked" 信号时会调用 hello() 函数，并将 NULL 传给
     * 它作为参数。hello() 函数在前面定义了。*/
    g_signal_connect (G_OBJECT (button), "clicked", G_CALLBACK (hello), NULL);
    /* 当单击按钮时，会通过调用 gtk_widget_destroy(window) 来关闭窗口。
     * "destroy" 信号会从这里或从窗口管理器发出。*/
   g_signal_connect_swapped(G_OBJECT(button),"clicked",
   G_CALLBACK(gtk_widget_destroy), window);
    /* 把按钮放入窗口(一个 Gtk 容器)中。*/
    gtk_container_add (GTK_CONTAINER(window), button);
    /* 最后一步是显示新创建的按钮和窗口 */
    gtk_widget_show (button);
    gtk_widget_show (window);
    /* 所有的 Gtk 程序必须有一个 gtk_main() 函数。程序运行停在这里
     * 等待事件 (如键盘事件或鼠标事件) 的发生。*/
    gtk_main ();
    return 0;
}
```

9.2　Gtk+用户界面测试工具 Gerd

Gerd 是 Gtk+开发的一个模块，它是开源的，并遵循 LGPL 协议。Gerd 实际上是一个录制器，它可以使程序员对任意运行的 Gtk+程序中的事件进行录制和回放。

目前的 Gerd 只能对 Gtk+ v1.2 开发的界面进行录制/回放，暂时不支持 Gtk+ v2.0。Gerd 的下载地址为 http://testbit.eu/~timj/historic/gerd/gerd-0.0.3.tar.gz。

9.2.1　Gerd 测试环境建立

下面以 Linux 平台为例，以 Gtk+ v1.2.10 开发的 GUI 为测试目标，安装 Gerd 工具。安装过程如下。

(1) 从网上下载 Gerd(gerd-0.0.3.tar.gz)的源文件。

(2) 解压缩文件至指定文件夹。

(3) 根据 readme 文档中的提示进行安装。首先，启动一终端，并将其路径设到解压到的文件夹下，运行“./configure”，进行系统适配。

在适配过程中，会提示如图 9-2 所示的问题。

```
root@localhost:~/gerd-0.0.3
文件(F)  编辑(E)  查看(V)  终端(T)  转到(G)  帮助(H)
creating libtool
updating cache ./config.cache
loading cache ./config.cache
checking for X... libraries /usr/X11R6/lib, headers /usr/X11R6/include
checking for dnet_ntoa in -ldnet... no
checking for dnet_ntoa in -ldnet_stub... no
checking for gethostbyname... no
checking for gethostbyname in -lnsl... no
checking for connect... no
checking for connect in -lsocket... no
checking for remove... no
checking for remove in -lposix... no
checking for shmat... no
checking for shmat in -lipc... no
checking for IceConnectionNumber in -lICE... no
checking for gtk-config... /usr/local/bin/gtk-config
checking for GTK - version >= 1.2.7... no
*** Could not run GTK test program, checking why...
*** The test program failed to compile or link. See the file config.log for the
*** exact error that occured. This usually means GTK was incorrectly installed
*** or that you have moved GTK since it was installed. In the latter case, you
*** may want to edit the gtk-config script: /usr/local/bin/gtk-config
configure: error: Cannot find GTK gtk-config missing from path?
[root@localhost gerd-0.0.3]#
```

图 9-2　Gerd 适配过程中的错误提示信息

按照提示信息，查看 config.log 文件，发现出错原因是当前版本的 GCC 不支持-fnonnull-objects 编译选项。在 configure 文件中删除该选项，如图 9-3 所示。

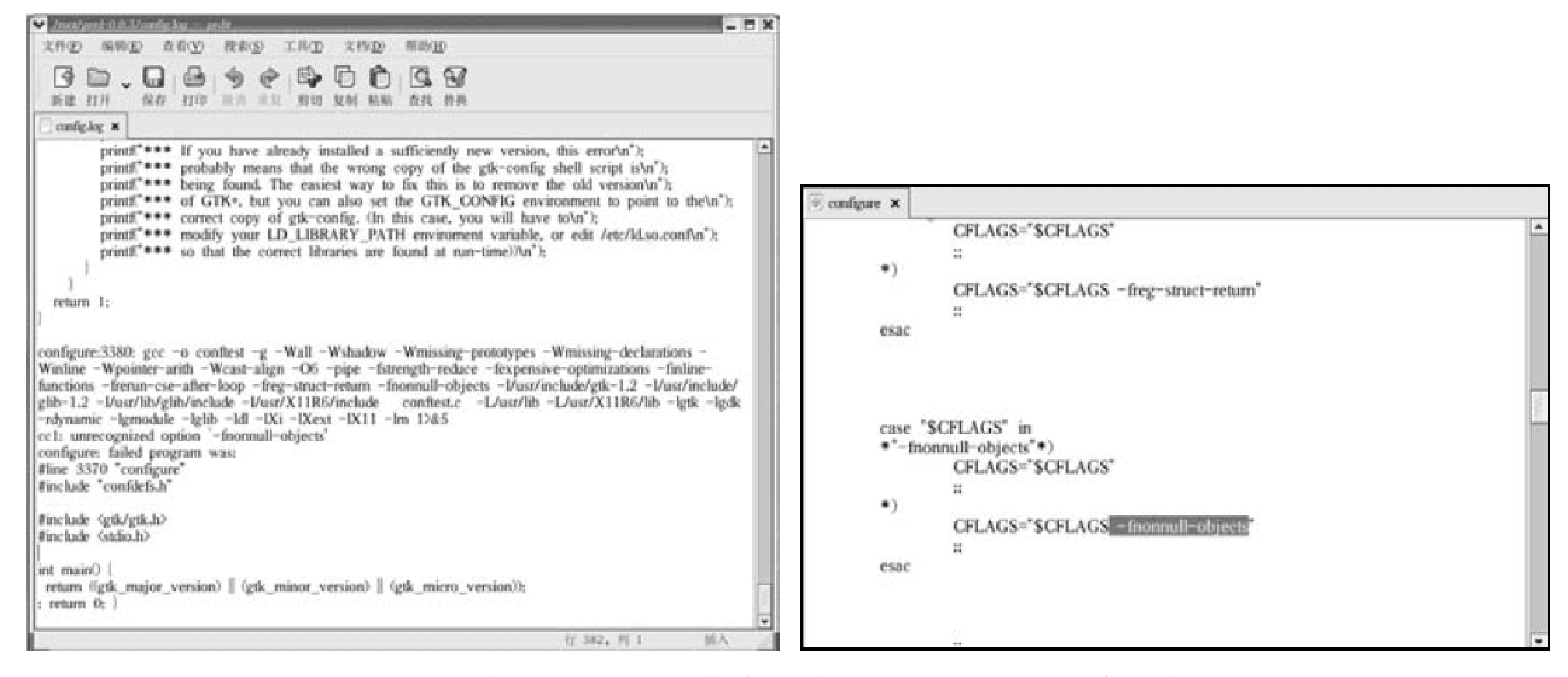


图 9-3　在 configure 文件中删除-fnonnull-objects 编译选项

(4) make & make install。

再次运行“./configure”，成功后，输入“make”和“make install”，就能够成功安装。

(5) 运行。

运行时会出现问题，提示缺少libgerd.so文件。进入/root/local/lib文件夹下，把与libgerd相关的三个文件均放到/root/lib下，即可解决问题，如图9-4所示。

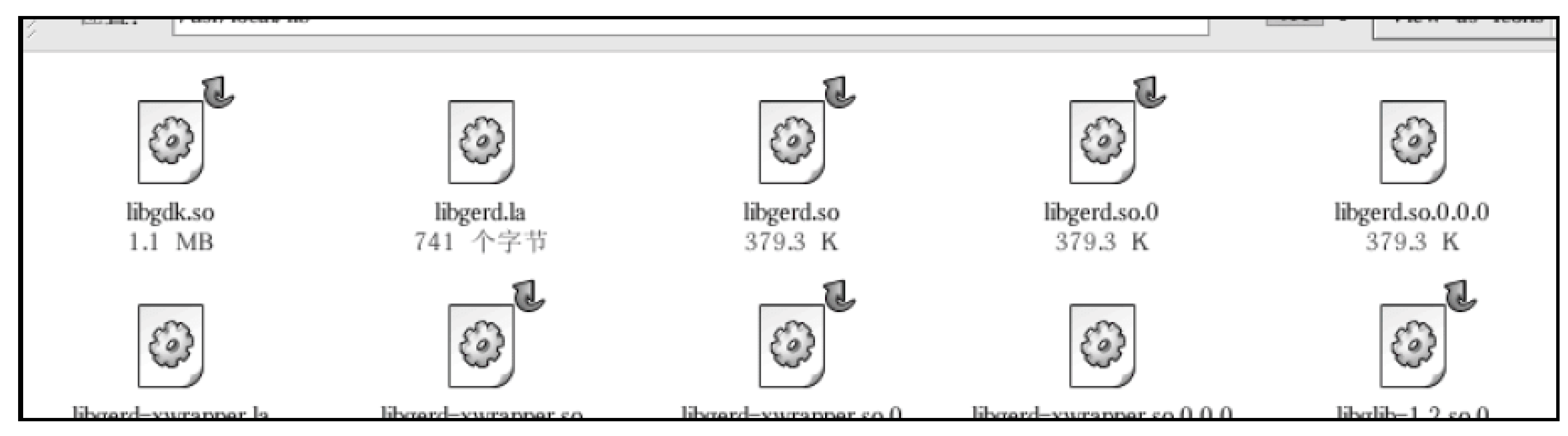


图9-4　对libgerd.*进行复制

9.2.2　Gerd功能及使用原理

Gerd是Gtk+ Event Recorder以及GTK+ module，它能记录下用户任意的Gtk+程序的操作，并将其中的事件重放。

使用Gerd录制用户的操作及重放功能可以方便有效地进行自动测试。

Gerd的主要功能可以在Gerd安装完毕后，输入“gerd”并按Tab键进行显示，如图9-5所示。

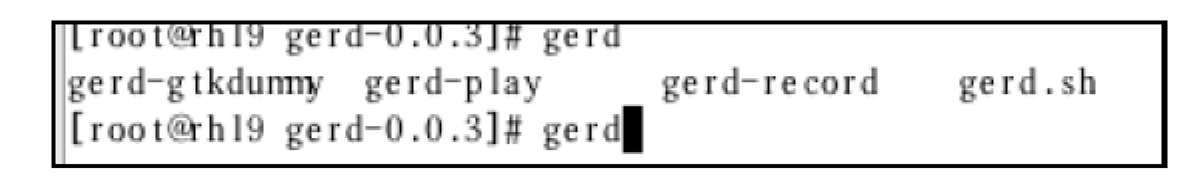


图9-5　显示Gerd的主要功能

其中主要使用的是命令gerd-play和gerd-record。

命令gerd-record的格式是:

```
gerd-record event-file application
```

其中application为被测程序；而event-file由用户指定，用来存储用户在application中所做的所有操作。该命令需运行在该软件目录下(可先设置好命令使用的路径)。

命令gerd-play的格式是：

```
gerd-play event-file application
```

其中application为被测程序；而event-file由用户指定，用来存储用户在application中所做的所有操作。该命令需运行在该软件目录下(也可先设置好命令使用的路径)。

在实际测试时，先运行gerd-record记录测试信息和详细步骤，之后想要进行自动测试时，运行gerd-play即可。

在 event-file 和 gerd-io.c 中，可以总结或归纳出 Gerd 所能识别和处理的几类事件。下面给出几个主要的事件类型。

Nothing：没有事件发生。

Delete：窗口管理程序送出删除窗口事件，该窗口将被删除。

Destroy：窗口已被撤销。

Expose：一个窗口有部分没有被覆盖。

NoExpose：一个窗口有部分没有被覆盖，但有关 Expose 的事件没有产生。

VisibilityNotify：一个窗口变成完全/部分/没有被遮挡。

MotionNotify：鼠标移动。

ButtonPress：鼠标按键。

ButtonRelease：鼠标松键。

KeyPress：键盘按键。

KeyRelease：键盘松键。

EnterNotify：进入一个窗口。

LeaveNotify：退出窗口。

FocusChange：窗口视点发生变化(视窗得到键盘事件)。

Resize：改变窗口尺寸。

Map：切换窗口(该窗口现在显示在屏幕上)。

Unmap：不切换窗口(该窗口不再显示在屏幕上)。

9.2.3　界面测试应用举例

下面以 Gtk+ v1.2.10 中的 testgtk 为例，来说明如何进行 Gtk+的 GUI 测试(可以在一个临时目录下安装 Gtk+ v1.2.10，这样就可以获得 testgtk 应用程序了)。

在 testgtk 软件的运行过程中，主要会使用到的操作有 Delete(关闭界面操作)、MotionNotify(鼠标移动操作)、ButtonPress(鼠标按键操作)、ButtonRelease(鼠标松键操作)、KeyPress(按键操作)、KeyRelease(按键松开操作)、Configure(进入界面操作)、EnterNotify(进入控件操作)和 LeaveNotify(离开控件操作)。

下面的例子是对录制脚本进行修改，如对 testgtk 中如图 9-6 所示界面中的 button box 和 buttons 两个按钮的单击顺序进行修改。

脚本中关于两个按钮的操作(GtkButton 1 给出的是第一个按钮)：

```
(event (BUTTON_PRESS "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 1|
+0" FALSE 125 12 0.5 0 0 16 1 Mouse 0xfedc))
(event (BUTTON_RELEASE "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 1|
+0" FALSE 125 12 0.5 0 0 272 1 Mouse 0xfedc)
```

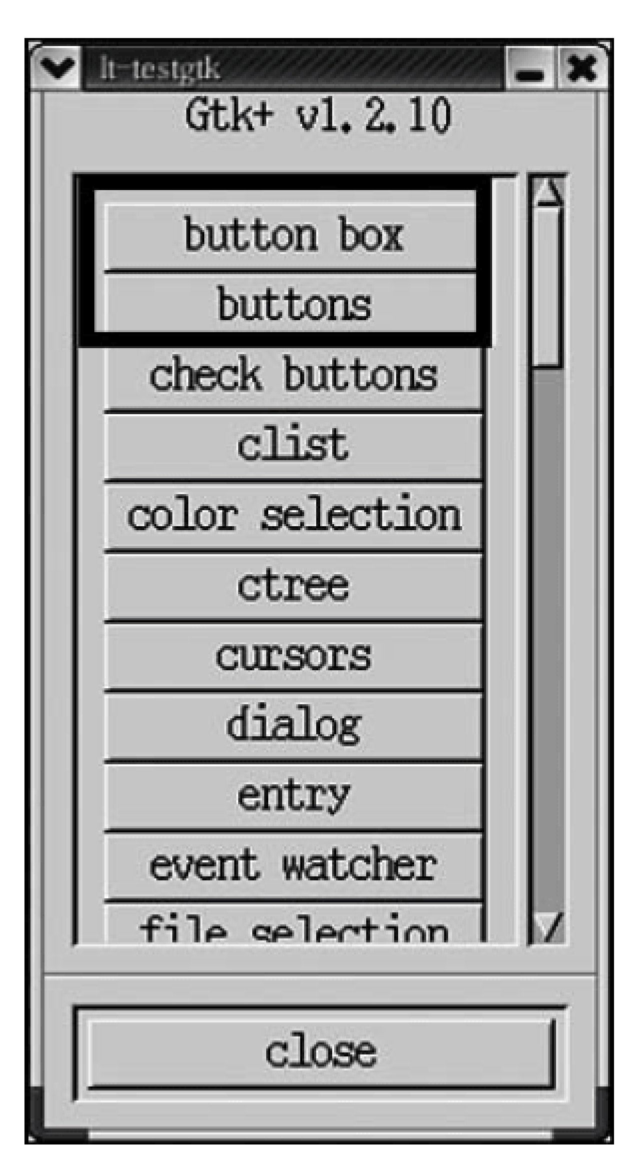


图 9-6　图中所显示的 button box 和 buttons

(event (DELETE "E:0001:%Tpl%Lt-testgtk%GtkWindow%lt-testgtk%Button Boxes%|+0" TRUE))

下面给出一些简单的脚本命令解释：

(event (CONFIGURE "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|+0" FALSE 6 19 200 400))

-- 是对整体的配置

(wait 0)

-- 指鼠标在控件上延迟的时间

(warp 168 318)

-- 鼠标在界面上的坐标

(event (ENTER "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 1|
+0" FALSE "N:0000:%NULL%" 136 17 Normal Anchestor TRUE 16))

-- ENTER 说明鼠标悬浮在 Gtk Button 1 上

(event (LEAVE "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 1|
+0" FALSE "N:0000:%NULL%" 125 12 Grab Anchestor TRUE 272))

-- LEAVE 说明鼠标从 Gtk Button 1 控件上离开

(event (BUTTON_PRESS "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 1|
+0" FALSE 125 12 0.5 0 0 16 1 Mouse 0xfedc))

-- 这里是对单击按钮进行操作

(event (BUTTON_RELEASE "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 1|
+0" FALSE 125 12 0.5 0 0 272 1 Mouse 0xfedc))

-- 这里是对释放按钮进行操作

```
(event (DELETE "E:0001:%Tpl%Lt-testgtk%GtkWindow%lt-testgtk%Button Boxes%|+0" TRUE))
-- 这里是单击“关闭”按钮的操作
```

请注意：在这个测试实例 testgtk 中(见图 9-6)，close 这个按钮和右上角的“×”这两个按钮的功能虽然一样，但是在脚本中 close 是一个名为 close 的按钮，而“×”是名为 DELETE 的一个操作。

1. 顺序颠倒

如果想对脚本中的两个按钮(GtkButton 1 和 GtkButton 2)的单击录制脚本顺序进行修改，那么只需要对脚本中的那两个对按钮进行操作的脚本颠倒顺序就可以实现。

2. 事件删除

```
(wait 0)
(warp 157 313)
(event (ENTER "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 2|
+0" FALSE "N:0000:%NULL%" 109 15 Ungrab Anchestor TRUE 272))
(wait 0)
(event (BUTTON_PRESS "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 2|
+0" FALSE 109 15 0.5 0 0 16 1 Mouse 0xfedc))
(wait 107)
(warp 157 313)
(event (BUTTON_RELEASE "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 2|
+0" FALSE 109 15 0.5 0 0 272 1 Mouse 0xfedc))
```

这段脚本说明了 Gtk Button 2 的工作情况，将它从脚本 1 中删除之后，第一个按钮的操作依然保留，第二个按钮的操作已经从脚本中删除了，所以没有显示。

3. 事件添加(增加了 GtkButton 3 的操作)

```
(wait 0)
(warp 188 313)
(event (ENTER "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 3|
+0" FALSE "N:0000:%NULL%" 109 15 Ungrab Anchestor TRUE 272))
(wait 0)
(event (BUTTON_PRESS "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 3|
+0" FALSE 109 15 0.5 0 0 16 1 Mouse 0xfedc))
(wait 107)
(warp 188 313)
```

```
(event (BUTTON_RELEASE "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 3|
+0" FALSE 109 15 0.5 0 0 272 1 Mouse 0xfedc))
```

这个时候添加脚本 Gtk Button 3 的操作，播放时就会显示增加了对第三个按钮的操作。

4. 时间延迟实验

```
(wait 1500)
(warp 157 313)
(event (ENTER "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|GtkVBox 1|
GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 3|
+0" FALSE "N:0000:%NULL%" 109 15 Ungrab Anchestor TRUE 272))
```

此时 Gtk Button 3 的单击将延迟 1.5s 后播放。

5. 事件更改

```
(wait 0)
(warp 157 313)
(event (ENTER "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|GtkVBox 1|
GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 2|
+0" FALSE "N:0000:%NULL%" 109 15 Ungrab Anchestor TRUE 272))
(wait 0)
(event (BUTTON_PRESS "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 2|
+0" FALSE 109 15 0.5 0 0 16 1 Mouse 0xfedc))
(wait 107)
(warp 157 313)
(event (BUTTON_RELEASE "E:0001:%Tpl%Lt-testgtk%main window%lt-testgtk%NULL-Title%|
GtkVBox 1|GtkScrolledWindow 2|GtkViewport 1|GtkVBox 1|GtkButton 2|
+0" FALSE 109 15 0.5 0 0 272 1 Mouse 0xfedc))
```

将录制文件中的 Gtk Button 2 改名为 Gtk Button 5，这时播放出来的结果是对 Gtk Button 2 的单击，但是弹出 Gtk Button 5 的窗口。

通过上面对测试脚本的修改或添加，可以实现图形用户界面的自动化测试。当然，从 Gerd 的脚本程序来看，确实很复杂；且由于资料缺乏，需要做大量的实验，详细地分析，才能清楚地理解 Gerd 脚本的语法和语义。对于其他图形用户界面的录制/回放工具来说，其脚本编程也不是一件轻松的事，这要求测试人员对被测程序有深入的理解并对脚本编程有丰富的经验，才能最终达到自己书写测试脚本以实现快捷的 GUI 自动化测试这一目标。

实 验 习 题

1. 在 Gtk+ v1.2.10 包的 examples 中选取 2～4 个例子，用 Gerd 进行界面操作录制，并修改其中的脚本程序，完成不同的操作功能，最后进行回放。

2. 学习其他界面测试工具(包括商用的)，总结这些工具的录制/回放功能及其测试方法。

第Ⅴ部分　性能测试篇

性能测试在软件的质量保证中起着重要的作用，它包括的测试内容丰富多样。中国软件评测中心将性能测试概括为三个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。通常情况下，三方面有效、合理地结合，可以实现对系统性能全面的分析和对瓶颈的预测。

应用在客户端性能的测试目的是考察客户端应用的性能，测试的入口是客户端。它主要包括并发性能测试、疲劳强度测试、大数据量测试和速度测试等，其中并发性能测试是重点。并发性能测试主要是进行负载测试和压力测试，其主要思路是：①以真实的业务为依据，选择有代表性的、关键的业务操作设计测试用例，以评价系统的当前性能；②当扩展应用程序的功能或者新的应用程序将要被部署时，通过负载测试来帮助确定系统是否还能够处理期望的用户负载，是否满足系统所要求的性能；③通过模拟成百上千个用户，重复执行和运行测试，可以确认性能瓶颈并优化和调整应用，目的在于寻找到瓶颈问题。

应用在网络上性能的测试重点是利用成熟先进的自动化技术进行网络应用性能监控、网络应用性能分析和网络预测。

对于应用在服务器上性能的测试，可以采用工具监控，也可以使用系统本身的监控命令。实施测试的目的是实现服务器设备、服务器操作系统、数据库系统、应用在服务器上性能的全面监控和优化。

性能测试目前基本上是靠工具来实现的，很难用手工来完成。性能测试工具通常是指那些用来支持压力/负载测试，能够用来录制和生成脚本、设置和部署场景、产生并发用户和向系统施加持续压力的工具。性能测试工具用得较多的是国外商用软件，如 QALoad、LoadRunner 等。最近几年，开源或共享的性能测试工具也逐渐在中小企业中应用起来，高等院校也将开源的性能测试工具作为实践教学的重要手段。

目前市场上的性能测试工具种类很多，可以简单地划分为以下几种：负载/压力测试工具、资源监控工具、故障定位工具以及调优工具。

1. 商用性能测试工具

负载性能测试工具的原理通常是通过录制/回放脚本、模拟多用户同时访问被测试系统，来制造负载，产生并记录各种性能指标，生成分析结果，从而完成性能测试的任务。主流的负载性能测试工具有以下几个。

1) QALoad

Compuware 公司的 QALoad 是客户/服务器系统、企业资源配置(ERP)和电子商务应用

的自动化负载测试工具。QALoad 是 QACenter 性能版的一部分，它通过可重复的、真实的测试来彻底地度量应用的可扩展性和性能。QACenter 汇集了完整的跨企业的自动测试产品，是专为提高软件质量而设计的。QACenter 可以在整个开发生命周期、跨越多种平台、自动执行测试任务。

2) SilkPerformer

一种在工业领域较高的企业级负载测试工具。它可以模拟成千上万的用户在多协议和多计算的环境下工作。不管企业电子商务应用的规模大小及复杂性，通过 SilkPerformer，均可以在部署前预测它的性能。可视化的用户界面、实时的性能监控和强大的管理报告，可以帮助用户迅速地解决问题。例如，加快产品投入市场的时间，通过最小的测试周期保证系统的可靠性，优化性能和确保应用的可扩充性等。

3) LoadRunner

一种较高规模适应性的自动化负载测试工具，它能预测系统行为、优化性能。LoadRunner 强调的是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测，来帮助人们更快地确认和查找问题。此外，LoadRunner 能支持最宽泛的协议和技术，为一些特殊环境量身定做地提供解决方案。

4) IBM Rational Performance Tester

Rational Performance Tester 自动化负载和性能测试工具，用于开发团队在部署基于 Web 的应用程序前验证其可扩展性和可靠性。它提供了可视化编辑器，使测试新手使用起来可以更简单；为需要高级分析和自定义选项的专家级测试人员提供了对丰富的测试详细信息进行访问的能力，并支持自定义 Java 代码插入；能自动检测和处理可变数据，以简化数据驱动的测试；提供有关性能、吞吐量和服务器资源的实时报告，以便及时发现系统的瓶颈；可以在 Linux 和 Windows 上进行测试录制和修改。Rational Performance Tester 还是一个团队在对复杂的电子商务应用进行部署之前对其可度量性和可靠性进行性能测试构建、执行和分析的工具。它全面、低干扰的记录技术能够捕获在 HTTP/HTTPS 或者基于 SQL 的协议中客户端与服务器之间的通信。另外，它的嵌入式数据相关性过滤器能够检查可变数据，并根据数据驱动加载测试需求进行测试。Rational Performance Tester 测试工作量日程安排是完全可定制的，能够对真实的用户进行高精度模拟。最后，它的实时报告能显示从用户到用户组的精确到秒的响应时间，一旦系统出现瓶颈就马上显示出来。

5) WebLoad

WebLoad 是 RadView 公司推出的一个性能测试和分析工具，Web 应用程序开发者能够应用它来自动执行压力测试。WebLoad 通过模拟真实用户的操作，生成压力负载来测试 Web 应用程序的性能。用户创建的是基于 JavaScript 的测试脚本，称为议程(agenda)，用它来模拟客户的行为，通过执行该脚本来衡量 Web 应用程序在真实环境下的性能。目前，WebLoad 有专业版和开源版两个版本。

2. 开源免费的性能测试工具

1) WAS

Microsoft Web Application Stress (WAS) Tool 是由微软网站测试人员开发，专门用来进行实际网站压力测试的一套工具。通过这套功能强大的压力测试工具，用户可以使用少量的客户端计算机来仿真大量用户上线时对网站服务所可能造成的影响。

2) JMeter

Apache JMeter 是 100%的 Java 桌面应用程序，它被设计用来加载被测软件的功能特性。度量被测试软件的性能。设计 JMeter 的初衷是测试 Web 应用，后来又扩充了其他的功能。JMeter 可以完成针对静态资源和动态资源(静态文件、CGI 脚本、Java 对象、数据库、FTP 服务器等)的性能测试。JMeter 可以模拟大量的服务器负载、网络负载、软件对象负载，通过不同的加载类型全面测试软件的性能。JMeter 还提供图形化的性能分析。

3) p-unit

p-unit 是一款开源的性能测试框架。和 JUnit 不同，JUnit 关注的是测试用例的正确性，而 p-unit 不仅关注测试用例的正确性，还收集测试用例的性能参数。默认情况下，p-unit 收集测试用例的时间和内存消耗情况，可以产生文件、图片以及 PDF 格式的报表。此外，p-unit 还支持参数化测试、多线程测试以及不同 Java 虚拟机性能之间的比较。

4) OpenSTA

OpenSTA 是专用于 B/S 结构的、免费的性能测试工具。除了具有免费、开源的优点外，它还能对录制的测试脚本按指定的语法进行编辑。测试工程师在录制完测试脚本后，只需要了解该脚本语言的特定语法知识，就可以对测试脚本进行编辑，以便再次执行性能测试时获得所需要的参数，之后再进行特定的性能指标分析。OpenSTA 以最简单的方式让用户对性能测试的原理有较深的了解，其较为丰富的图形化测试结果也大大提高了测试报告的可阅读性。

OpenSTA 采用了基于 Common Object Request Broker Architecture (CORBA)的结构体系。它通过虚拟一个 proxy，使用其专用的脚本控制语言，来记录通过 proxy 的所有 HTTP traffic。测试工程师通过分析 OpenSTA 的性能指标收集器所收集的各项性能指标，以及 HTTP 数据，来对被测试系统的性能进行分析。

5) TPTP

Eclipse Test and Performance Tools Platform(TPTP)可以监测运行的并发线程数据、内存的使用情况等，是一款非常不错的性能测试工具。它是 Eclipse 官方的一款插件项目，可用来进行程序执行时间的统计分析、内存的监控、对象调用的分析等。

第10章　单元性能测试

随着网络的发展，软件也越来越复杂，从独立的单机结构到 C/S 结构、B/S 结构、多层体系架构、面向服务(SOA)结构等，集成的软件技术越来越多，支持的软件用户也越来越多。一个凸显在人们面前的问题便是性能问题。很多软件系统在开发测试时没有任何问题，但是上线不久就崩溃了，原因就在于缺少了性能方面的验证。因此，对于这类软件，其性能测试应该“从小做起”，从单元做起。

10.1　单元性能测试概念介绍

软件是否在上线之前进行性能测试就能解决问题呢？不一定，如果性能测试进行得太晚，会带来修改上的风险。很多软件系统在设计的时候并没有很好地考虑性能问题和优化方案，等到整个软件系统开发出来后，测试人员忙着集成测试，开发人员也疲于应付发现的功能上的 Bug，当所有功能上的问题都得到解决后，才想到要进行性能测试。而性能测试结果表明系统存在严重的问题，如响应时间迟缓、内存占用过多、不能支持大量的数据请求、在大量用户并发访问的情况下会造成系统崩溃等。此时再去修改程序就已经非常困难了，因为要彻底地解决性能问题，就需要重新调整系统的架构设计，大量的代码需要重构，这时的程序员已经筋疲力尽，不想再进行代码的调整了，因为调整带来的是大量的编码工作，同时可能引发大量的功能上的不稳定性和再次出现大量的 Bug。

这给测试人员一个启示，性能测试不应该只是一种后期的测试活动，更不应该是软件系统上线前才进行的“演练”，而应该贯穿软件开发的全过程，如图 10-1 所示。

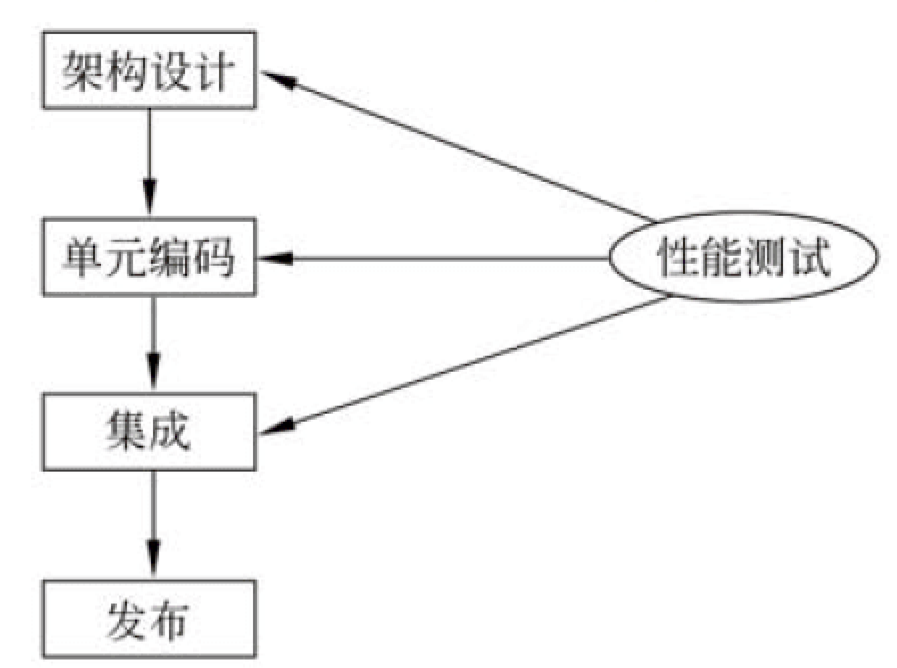


图 10-1　贯穿软件开发全过程的性能测试

对于性能的考虑应该在架构设计时就开始，对于架构原型要进行充分的评审和验证。由于架构设计是一个软件系统的基础平台，如果基础不好，也就是根基不牢，那么性能问题就会根深蒂固，后患无穷。

性能测试应该在单元测试阶段就开始。从代码的每一行效率，到一个方法的执行效率，再到一个逻辑实现的算法效率；从代码的效率，到存储过程的效率，都应该进行优化。单元阶段的性能测试可以考虑从以下几个方面进行：代码效率评估，应用单元性能测试工具，操作系统、数据库及网络等的优化。

应该注意每一行代码的效率，所谓“积少成多，水滴石穿”。一些看似细小的问题可以经过多次的执行累积成一个大的问题，特别是在一个庞大的系统中，经过多次的调用，问题会逐渐地被放大，直到发生从量变到质变的爆发。这些问题都可以通过代码走查来发现。

测试人员可以使用一些代码效率测试工具来帮助找出哪些代码或方法在执行时需要耗费比较长的时间。例如，AQTime 就是一款可以计算出每行代码执行时间的工具，它可以给出每一个方法甚至每一行代码的执行时间是多少，这对开发人员在查找代码层的性能瓶颈时会有很大的帮助。

除了代码行效率测试工具外，最近还出现了一些开源的单元级别的性能测试框架，可以像使用 xUnit 这一类的单元测试框架一样，但它们不是用于测试单元代码的正确性，而是用于测试函数、方法的性能是否满足要求。例如，NTime 就是这样的一个小工具。

NTime 可以并发地运行同一个方法多次，查看能否达到预期的性能指标。例如，下面的代码使用 NTime 框架启动两个线程，在 1s 内并发地执行 MyTest 方法多次。

```
[TimerHitCountTest(98,Threads = 2,Unit = TimePeriod.Second)]
Public void MyTest()
{
    //调用被测试的方法
    MethodToBeTest();
}
```

如果测试结果表明能执行超过 98 次，则认为 MethodToBeTest 方法的性能达标，否则将被视为不能满足性能要求。

然而，做好单元性能测试绝非易事。《代码大全》的作者 Steve McConnell 曾经说过“不要过早的优化，也不要过早的劣化”，Donald Knuth也有过类似的言论：“过早优化是万恶之源”。很多软件工程专家也支持这样的观点，他们始终认为过早的优化会带来以下一些问题。

(1) 过早地关注不重要的部分而忽略了行动目标本身，这在实践中往往导致为了妥协性能而丢失了系统关键功能的本末倒置的现象发生。

(2) 程序本身是在演变的，过早地关注和采集性能指标对于系统的状况经常起不到指导作用。

(3) 早期的性能测试往往是在开发环境中进行的，开发环境中采集的性能数据不能代

表这段程序在实际运行环境中的情况，容易导致系统性能本来达标，但是由于过早在开发环境中采集到了较低的性能数据而做无谓的优化，甚至破坏原有功能的风险。

那么，是不是早期就不该关注优化问题呢？正如前面说到的，如果当性能问题出现在上线之前，那时将已经很难修改，系统面临失败。如何才能很好地权衡优化的时机和粒度呢？这个问题一直是性能专家们所探讨的核心话题。一般来说，关注性能问题应该分两步走：①根据早期的需求采集数据，估算出访问量的量级，通过这个量级选择合适的架构，如果未来性能问题出现在架构上，那么这将意味着天翻地覆的修改，所以这个分析工作很重要；②就开发者所开发的业务性代码的性能问题而言，这部分因为开发者水平参差不齐，很难有一种统一的约束，所以一个好的单元性能测试工具将帮助开发人员捕捉微小的性能问题。

单元性能测试和单元测试类似，是对系统运行的最小单元进行测试，不过单元性能测试更关注的是运行单元的性能指标。

不过早的优化并不意味着不过早地关注性能话题，单元性能测试工具可以持续捕捉程序演变到每个阶段的性能参数，可以选择在开发环境中运行，也可以选择在服务器环境中运行，并根据项目情况在需要的时候进行优化。

下面将介绍一款国内生产的单元性能测试工具 p-unit。

10.2　单元性能测试工具 p-unit

前面章节用大量篇幅介绍了使用 JUnit 进行单元测试来保证代码质量，但在实际中可能会经常碰到下面这样的问题：①程序在多线程下的正确性如何？性能如何？②对于 Java 程序来说，虽然它有垃圾收集机制，但是两个不同的 Java 程序员实现相同功能的 Java 程序可能使用的内存大不相同。仅这两点，就很难通过 JUnit 来解决，其他的就更不用说了。而 p-unit 就能够很好地解决这些问题。如：

(1) 同一个测试用例，可以单线程运行，也可以多线程运行。和 JUnit 不同，p-unit 的思想是测试用例就是测试用例，不牵涉运行逻辑，所以同一个测试用例，可以被不同的 CPU 运行。p-unit 测试用例无须实现任何接口，所以当然能兼容 JUnit 测试用例，包括 setUp()/tearDown()函数。

(2) 对于每个测试函数，p-unit 都给出了运行时间和内存消耗情况。这使得进行单元测试的同时，也做了时间/内存的性能测试，而无须到出现性能瓶颈时再使用昂贵的商用性能测试工具来查找问题。

(3) p-unit 尤其侧重性能的测试，它支持同一个测试用例在不同参数下的运行。相信有性能测试经验的程序员都知道，同一个测试场景，往往要测试三四个数量级的不同数据，检查性能是线性增长还是几何增长。这对性能评估非常重要。

测试结果方面，p-unit 非常灵活，默认的形式有控制台输出、文件输出、图片和 PDF

报表。当以正确性测试为主的时候，可以选择控制台输出；而当以性能测试为主的时候，强烈建议选择图片和 PDF 输出，以便非常直观地看出每个测试用例所消耗的时间和内存情况。

p-unit 甚至支持不同虚拟机之间的性能比较，其结果反映到同一张图上，对比非常明显。随着 Java 的开源，现在虚拟机的选择也越来越多，Sun、BEA、IBM、Apache Harmony、Kaffe，哪一台虚拟机最适合程序，p-unit 将非常容易地给出答案。

10.2.1　p-unit 测试环境建立

p-unit 是一款 SourceForge 上的国产开源单元性能测试工具，构建 p-unit 测试环境总共包含以下一些环境。

(1) 选择一款集成开发环境。一个好的开发环境能使开发和测试工作更加便利，在本例中，选择最广为人知的 Eclipse 作为开发和测试环境。

(2) 获得 p-unit 开发包。从 p-unit 官方主页 http://sourceforge.net/projects/p-unit 下载。下载最新的 p-unit-0.15-all，它是一个 zip 文件，解压后里面包含核心组件 p-unit-0.15.319.jar 和 p-unit-0.15.319-extension.jar，figures 目录中存放的是 p-unit 所能导出的三种报告，libs 目录中有效的是 p-unit 所依赖的第三方包，punit.samples 目录中是一个 p-unit 的例子。

(3) 配置开发环境。在 Eclipse 中创建 Java 新项目 punit-test，在该项目中创建三个目录：bin、src 和 libs。将 zip 包中的 p-unit-0.15.319.jar、p-unit-0.15.319-extension.jar 和 libs 下的所有 jar 文件复制到项目的 libs 目录下。选择所有 jar 文件，右击后选择 Build Path 命令，然后就可以在 src 目录下编写测试用例了，如图 10-2 所示。

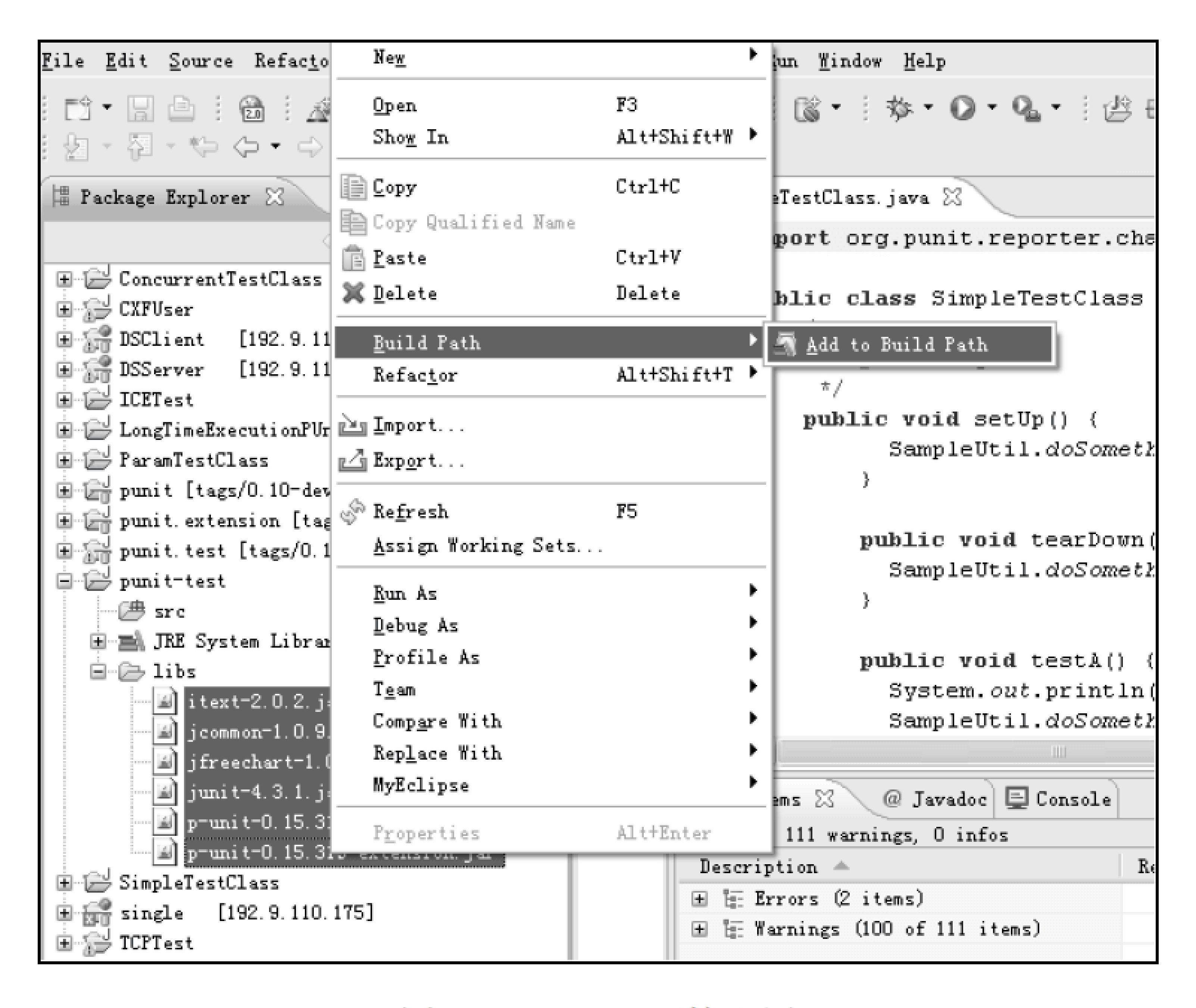


图 10-2　p-unit 环境配置

10.2.2　p-unit 测试功能及使用流程

p-unit 测试功能主要包括以下几个。

(1) 单线程测试：对单一线程程序进行测试，收集运行时间、内存占用情况等数据。

(2) 多线程测试：p-unit 同时还支持捕捉多线程并发执行时每个线程的运行时间和内存占用情况，同一个测试用例可以单线程执行，也可以多线程执行，测试用例开发者只需写一套测试用例即可。

(3) 参数化测试：方便测试同一个线程在不同数据级上的性能表现。

(4) 多虚拟机测试：只需指定虚拟机路径，就能够方便地测试同一个线程在不同虚拟机上的性能表现，报表上可以非常直观地显示性能差别(可惜的是在最新版本的 p-unit 中取消了对多虚拟机测试的支持)。

(5) 与 JUnit 兼容：最新版本的 p-unit 还能同时运行早期的 JUnit 测试用例。

(6) 像 JUnit 一样，p-unit 也可以定义自己的 testSuite，对 testSuite 中的一批 testCase 进行统一测试。

(7) p-unit 支持一个缓冲池，当同时运行的测试用例太多的时候，使用缓冲池来限制最多同时运行的数量，而让其他的处于等待状态。

除了上述这些测试功能之外，p-unit 还采用事件处理机制来将测试结果装载到不同的介质中，在最新版本的 p-unit 中支持三种测试结果存储介质：JPG 图表、txt 文档和 PDF 文件。

使用 p-unit 进行测试的一般流程如下。

(1) 创建运行器。在 p-unit 中主要包含单线程运行器 SoloRunner 和多线程运行器 ConcurrentRunner。

(2) 为运行器添加监听事件。在 p-unit 中事件主要包括文件记录事件 FileLogger、图像记录事件 ImageRender 和 PDF 记录事件 PDFRender。

(3) 把测试用例传入运行器中运行。其中，测试用例的 setUp()方法在所有测试执行前执行，tearDown()方法在所有测试执行后执行，以“test”开头的方法将会被执行，最终将执行结果生成指定的文件。

10.2.3　p-unit 测试应用举例

仍以前面章节介绍的自动售货机为例。下面是被测程序代码：

```
package unit_test;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
```

```
import java.util.Scanner;
public class CoinMachine
{
  long i = 0L;
  long n = 0L;
  long change = 0L;
  public static void main(String[] args)
    throws IOException
  {
    CoinMachine coinmachine = new CoinMachine();
    System.out.println("Welcome to the vending machine!\n Orange(0.5$)\n Beer(0.5$)");
    for (String c = "y"; (c.equals("y")) || (c.equals("Y")); ) {
      coinmachine.decde(coinmachine.coin(), coinmachine.goods());
      System.out.println("Continue?(Y/N) \n");
      BufferedReader user = new BufferedReader(new InputStreamReader(System.in));
      c = user.readLine();
    }
  }
  public long coin()
  {
    for (this.i = 0L; this.i == 0L; ) {
      System.out.println("Please choose the type of coin:  \n 1. 0.5$ \n 2. 1.0$ \n");
      Scanner s = new Scanner(System.in);
      this.i = s.nextInt();
      if ((this.i != 1L) && (this.i != 2L)) {
        this.i = 0L;
        System.out.println("Error!Please try again!\n");
      }
      else if (this.i == 1L) {
        this.i = 5L;
      } else {
        this.i = 10L;
      }
    }
    return this.i;
  }
  public long goods()
  {
    for (this.n = 0L; this.n == 0L; ) {
      System.out.println("Plese choose the goods:  \n 1.Orange \n 2.Beer \n");
      Scanner s = new Scanner(System.in);
```

```
            this.n = s.nextInt();
            if ((this.n != 1L) && (this.n != 2L)) {
                this.n = 0L;
                System.out.println("Error!Please try again!\n");
            }
        }
        return this.n;
    }
    public String decde(long x, long y)
    {
        String s = "";
        if ((this.change = = 0L) && (x == 10L)) {
            s = "No changes left,can not continue!Retrieve coins!";
        }
        else if ((x = = 5L) && (y == 1L)) {
            s = "Here is your Orange!";
            this.change += 1L;
        }
        else if ((x == 5L) && (y == 2L)) {
            s = "Here is your Beer!";
            this.change += 1L;
        }
        else if ((x == 10L) && (y == 1L)) {
            s = "Here are your Orange and changes(0.5$)!";
            this.change -= 1L;
        }
        else if ((x == 10L) && (y == 2L)) {
            s = "Here are your Orange and changes(0.5$)!";
            this.change -= 1L;
        }
        System.out.println(s);
        return s;
    }
}
```

下面给出上述代码 Coinmachine 类中 decde()方法的 JUnit 测试用例(测试类为 CoinMachineTest)：

```
package unit_test;
import junit.framework.TestCase;
public class CoinMachineTest extends TestCase {
        private CoinMachine coinmachine;
```

```
        long change=0;
        protected void setUp() throws Exception {
            super.setUp();
            coinmachine = new CoinMachine();
        }
        protected void tearDown() throws Exception {
            super.tearDown();
        }
        public void testDecde1() {
            assertEquals(coinmachine.decde(10, 1),"No changes left,can not continue!Retrieve coins!");
        }
        public void testDecde2() {
            assertEquals(coinmachine.decde(10, 2),"No changes left,can not continue!Retrieve coins!");
        }
        public void testDecde3() {
            assertEquals(coinmachine.decde(5, 1),"Here is your Orange juice!");
        }
        public void testDecde4() {
            assertEquals(coinmachine.decde(5, 2),"Here is your Beer!");
        }
    }
```

JUnit 下的测试结果表明没有问题，都是正确的，如图 10-3 所示。



图 10-3　JUnit 下 CoinMachine 类中 decde()方法的测试结果

接下来用 p-unit 对其进行性能测试(p-unit 测试用例无须继承任何测试类或实现接口，即可执行 test 开始的方法。尽管 JUnit 4 中加入了注释特性，但测试方法前缀为“test”仍然是测试人员的约定。因此如果 JUnit 测试用例遵循的是 test 命名规则，那么 p-unit 可以兼容运行 JUnit 测试用例)：

```
package unit_test;
import org.punit.reporter.chart.OverviewReporter;
import org.punit.reporter.chart.image.ImageRender;
import org.punit.reporter.chart.pdf.PDFRender;
import org.punit.reporter.stream.file.FileLogger;
import org.punit.runner.ConcurrentRunner; //import org.jfree.chart.imagemap.*;
import org.punit.runner.SoloRunner; //import test_performance.ParamTestClass;
public class Test_main {
```

```
        public static void main(String[] args) {
            SoloRunner runner = new SoloRunner();
            runner.addEventListener(new OverviewReporter(new ImageRender()));
            runner.addEventListener(new OverviewReporter(new PDFRender()));
            runner.addEventListener(new FileLogger());
            runner.run(CoinMachineTest.class);
        }
    }
```

可以看到测试的性能结果是：

```
[solo] Starting unit_test.CoinMachineTest
unit_test.CoinMachineTest
No changes left,can not continue!Retrieve coins!
testDecde1() - [0.2288ms]
testDecde2() - [0.110908ms]
No changes left,can not continue!Retrieve coins!
Here is your Orange juice!
testDecde3() - [0.090794ms]
Here is your Beer!
testDecde4() - [0.107835ms]
total: 4, failures:0 (GREEN) - 1875.985635ms
```

从以上结果中可以看到执行测试的时间，保存以便与其他程序做对比，而且更加容易判断程序的时间性能。

1. 多线程执行案例

同一个测试用例，用户可以选择不同的测试环境去运行，而不是绑定在某一个特定的测试软件工具上。在 p-unit 中运行测试用例时，单线程执行只需要加入一行代码即可：

```
new SoloRunner().run(CoinMachieTest.class);
```

但是，随着多核计算机的普及，一些“单核”软件往往会成为性能的瓶颈，传统的单元测试软件不具备并发执行的能力。

p-unit 充分利用了多核的特性，能并行执行测试用例，这极大地提高了测试速度。配置多线程执行测试用例时，无须改动代码，只需增加一行代码即可：

```
runner.setExecutorPool(new ExecutorPool(5));
```

上面的代码表示将启动 5 个线程的线程池来并行执行测试用例。需要注意的是，由于是并行执行，所以测试用例必须是独立的，粒度为 class，即执行 TestClassA 时不会影响到同时执行的 TestClassB。由于是单元测试，因此建议关闭内存检测功能，以充分享受多核的速度。

如果要运行多个线程，请将 soloRunner 改为 ConcurrentRunner：

```
new   ConcurrentRunner().run(CoinMachieTest.class);
```

默认情况下，p-unit 启动 10 个线程来执行。要指定不同的线程数，只需将线程数作为参数传入 ConcurrentRunner 即可，如 New ConcurrentRunner(5).run(CoinMachineTest.class)。这样就可以执行 5 个线程了。

p-unit 甚至支持不同的测试用例有不同的线程数，这要求测试用例实现 p-unit 中定义的 Concurrent 接口，该接口的定义为：

```
public interface Concurrent {
    public int concurrentCount();
}
```

2. 参数化执行用例

性能测试，不同于单元测试，它经常要求测试不同数量级在同一个测试场景中的表现，JUnit 是一款非常优秀的单元测试工具，但没覆盖到这个方面。比如，要比较类库 Foo1 的方法 bar()和类库 Foo2 的方法 bar()哪个更符合自己的应用程序，则需要测试该函数在应用程序可能的数量级范围内的表现。有经验的开发者知道经常会碰到小数量级 A 好，而大数量级 B 可能更好的局面，因此全面的测试对于代码的性能理解非常重要，能帮助开发者做出正确的决定。p-unit 支持将参数传给测试方法，测试用例需要实现 p-unit 的 parameterizable 接口，该接口的主要方法是返回一组参数列表，这组列表的参数将会被一一传给测试方法。

下面针对被测试类 SampleUtil：

```
package param_test;
import java.util.Random;
import org.punit.util.ThreadUtil;
public class SampleUtil {
        private static Random _random = new Random();
        public static void consumeMemory(int length) {
                byte[] data = new byte[length];
                for(int i = 0, j = 0; i < data.length; ++i) {
                    ++j;
                }
        }
        public static void consumeTime(int time) {
                ThreadUtil.sleepIgnoreInterruption(time);
        }
        public static void doSomething() {
                consumeTime(Math.abs(_random.nextInt()) % 500);
```

```
            consumeMemory(Math.abs(_random.nextInt()) % 100000);
        }
    }
```

参数化测试用例设计如下：类 ParamTestClass。

```
package param_test;
//import org.punit.reporter.chart.OverviewReporter;
//import org.punit.reporter.chart.image.ImageRender;
//import org.punit.reporter.chart.pdf.PDFRender;
//import org.punit.reporter.stream.file.FileLogger;
//import org.punit.runner.SoloRunner;
import org.punit.type.Parameter;
import org.punit.type.Parameterized;
public class ParamTestClass implements Parameterized {
    public Parameter[] parameters() {
        return new Parameter[] {
            new ParameterImpl(10),
            new ParameterImpl(20),
            new ParameterImpl(30),
            new ParameterImpl(40)
        };
    }
    public void testA(ParameterImpl param) {
        SampleUtil.doSomething();
    }
    public void testB(ParameterImpl param) {
        SampleUtil.doSomething();
    }
    public void testC(ParameterImpl param) {
        SampleUtil.doSomething();
    }
    public void setUpAfterWatchers(Parameter param) throws Exception {
    }
    public void setUpBeforeWatchers(Parameter param) throws Exception {
    }
    public void tearDownAfterWatchers(Parameter param) throws Exception {
    }
    public void tearDownBeforeWatchers(Parameter param) throws Exception {
    }
    static class ParameterImpl implements Parameter {
        private int _count;
```

```
            ParameterImpl(int count) {
                _count = count;
            }
            public int count() {
                return _count;
            }
            public String toString() {
                return String.valueOf(_count);
            }
        }
    }
```

这个类显然要比自动售货机的类要复杂，涉及一次次的循环，增加了程序的复杂性。因此，时间上肯定要比第一个用得多，请看测试结果：

```
    [solo] Starting param_test.ParamTestClass
    param_test.ParamTestClass
    testA(10) - [365.00896ms]
    testA(20) - [345.736399ms]
    testA(30) - [440.435028ms]
    testA(40) - [395.91205ms]
    testB(10) - [359.998268ms]
    testB(20) - [29.506264ms]
    testB(30) - [400.505651ms]
    testB(40) - [462.185557ms]
    testC(10) - [411.367925ms]
    testC(20) - [437.441351ms]
    testC(30) - [272.843895ms]
    testC(40) - [437.440792ms]
total: 12, failures:0 (GREEN) - 6113.791278ms
```

3. 不同运行环境性能测试

随着 Java 的开源，出现了更多的 Java 运行环境，除了 Sun 的参考实现外，BEA、IBM 均有自己的 Java 运行环境，更有诸如 Apache Harmony 这样的开源运行环境(尽管现在 Apache Harmony 尚不能称为 Java 运行环境)。运行环境测试用例，为运行环境开发者以及选择运行环境提供了一定的帮助。比如下面的例子就是测试 java.util.ArrayList 和 java.util.Vector 在两个不同运行环境中的表现。测试用例的写法和普通的测试用例完全一样，只需告诉 p-unit 在不同运行环境下的 Java 路径及正确的 classpath，然后调用 runVMs() 函数即可：

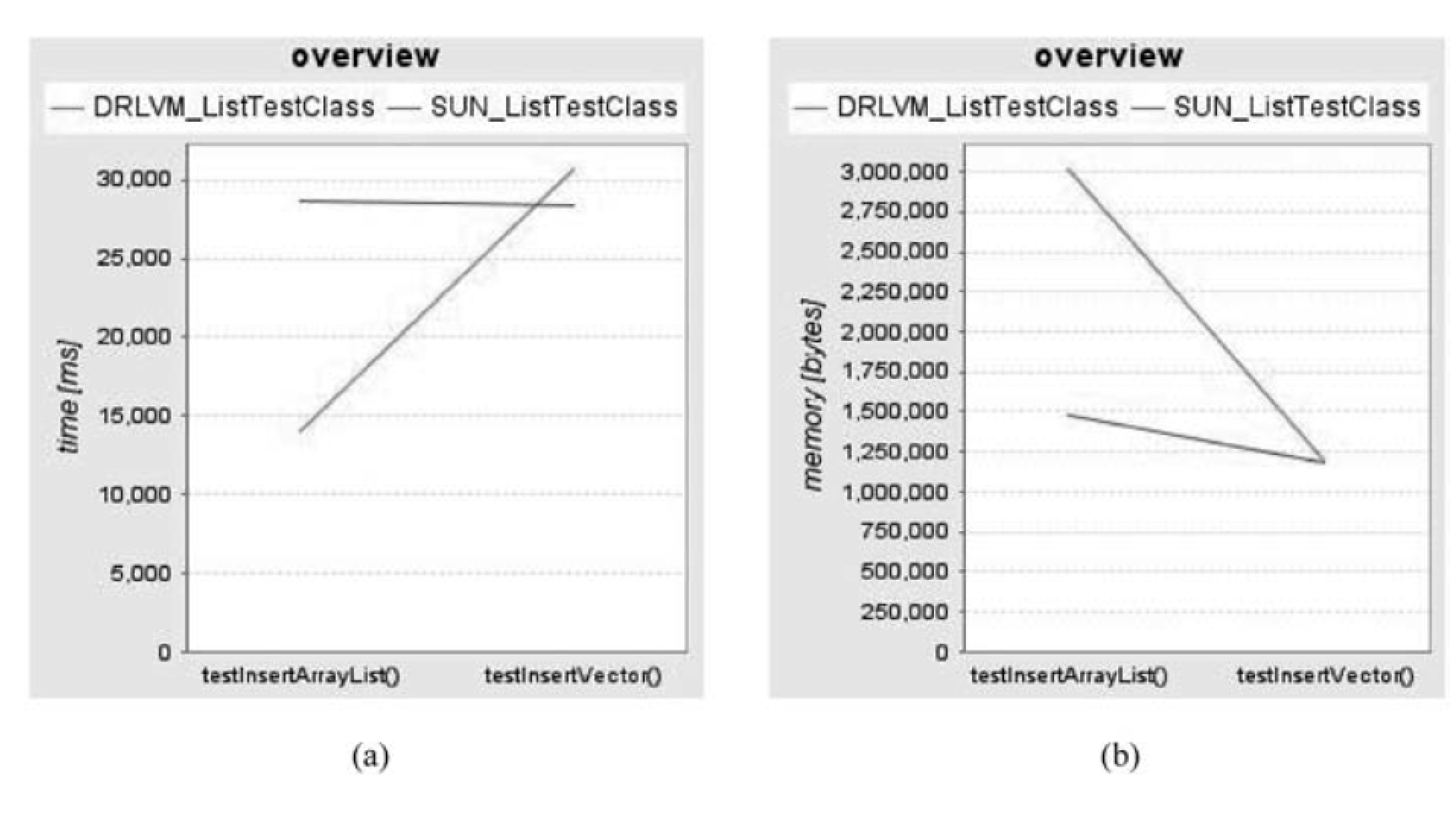
```
public static void main(String[] args) {
```

```
        PUnitSoloRunner runner = new PUnitSoloRunner();
        runner.addPUnitEventListener(new OverviewReporter(new ImageRender()));
        runner.runVMs(ListTestClass.class, new VM[] { VMConfig.HARMONY, VMConfig.SUN });
}
public class VMConfig {
        private static String CLASSPATH = " -cp correct_classpath_including_all_jars_and_path";
        private static String HARMONY_PATH = "harmony_path\\bin\\java" + CLASSPATH;
        private static String SUN_PATH = "sun_path\\bin\\java" + CLASSPATH;
        public static VM HARMONY = new VM(HARMONY_PATH, "HARMONY");
        public static VM SUN = new VM(SUN_PATH, "SUN");
}
```

运行环境测试类 ListTestClass：

```
public class ListTestClass {
    private static final int LIST_COUNT = 100000;
    private static Object element = new Object();
    private Random indexGenerator = new Random();;
    public void testInsertArrayList() {
        ArrayList arrayList = new ArrayList(LIST_COUNT);
        insertSequence(arrayList);
        insertRandom(arrayList);
    }
    public void testInsertVector() {
      Vector vector = new Vector(LIST_COUNT);
      insertSequence(vector);
      insertRandom(vector);
    }
    public void insertSequence(List list) {
        for (int i = 0; i < LIST_COUNT; ++i) {
            list.add(element);
        }
    }
    public void insertRandom(List list) {
        for (int i = 0; i < LIST_COUNT; ++i) {
            list.add(indexGenerator .nextInt(LIST_COUNT),element);
        }
    }
}
```

运行结果如图 10-4 所示。



(a)　　　　(b)

图 10-4　运行环境测试用例的运行结果

从图中可以很直观地看出，所使用的 Harmony 版本在该测试案例中速度更快(图 10-4(a))，但内存消耗更多(图 10-4(b))。下面介绍如何输出报表，但或许读者已经注意到了，代码非常简单。

4. 事件机制构架

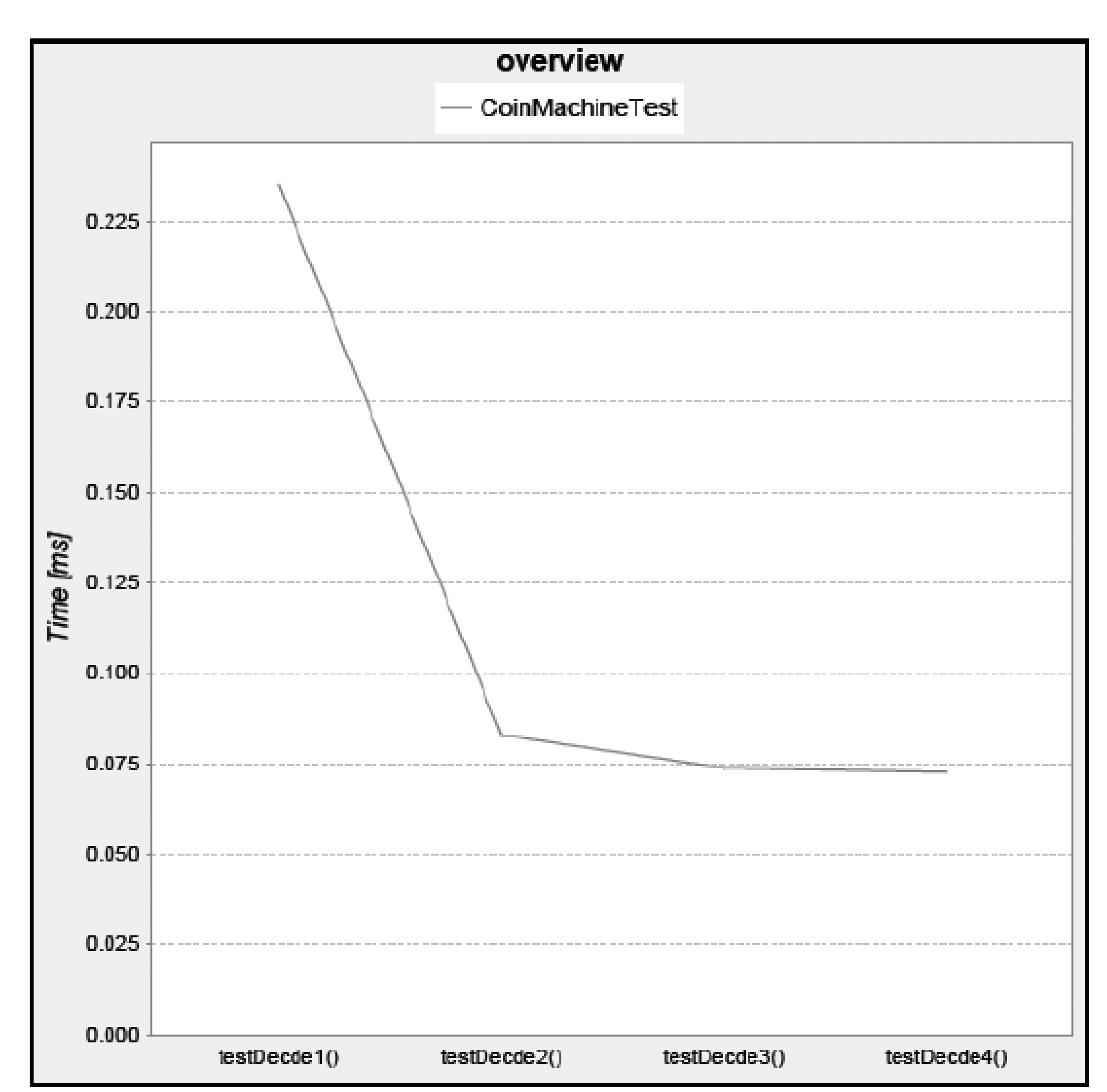
我们已经看到过 p-unit 输出结果的两种形式：控制台和图片报表。默认情况下，p-unit 将输出到控制台。p-unit 采用事件机制，在运行器的每个节点都会提供通知事件。所有的输出都是通过注册事件响应器来实现的。这也表明了结果输出和运行器完全隔离，用户也可以定制自己的报表。p-unit 有 4 种内建输出，分别为控制台、文件、图片报表和 PDF 报表。

p-unit 内建的报表分为三种不同的粒度：总体级别(OverviewReporter)、TestSutie 级别(TestSuiteReporter)和测试用例类级别(TestClassReporter)。这三种级别都可以输出图片格式或 PDF 格式，因此总共有 6 种类型的输出。上述代码就是输出总体级别的图片。由于事件监听器是互相独立的，因此既可以选择输出图片又可以选择输出 PDF 文件，只需再添加事件监听器即可：

```
runner.addEventListener(new OverviewReporter(new ImageRender()));
runner.addEventListener(new OverviewReporter(new PDFRender()));
```

程序运行之后，会自动产生一个名为 result 的文件夹，里面包含产生的图形结果——图片或 PDF 格式的显示结果，如图 10-5 所示。

为了比较几个不同参数的测试用例的执行结果，编写了三个不同的类来比较运行时间的不同，以测试各个单元的性能，以及使用可视化的结果进行分析。



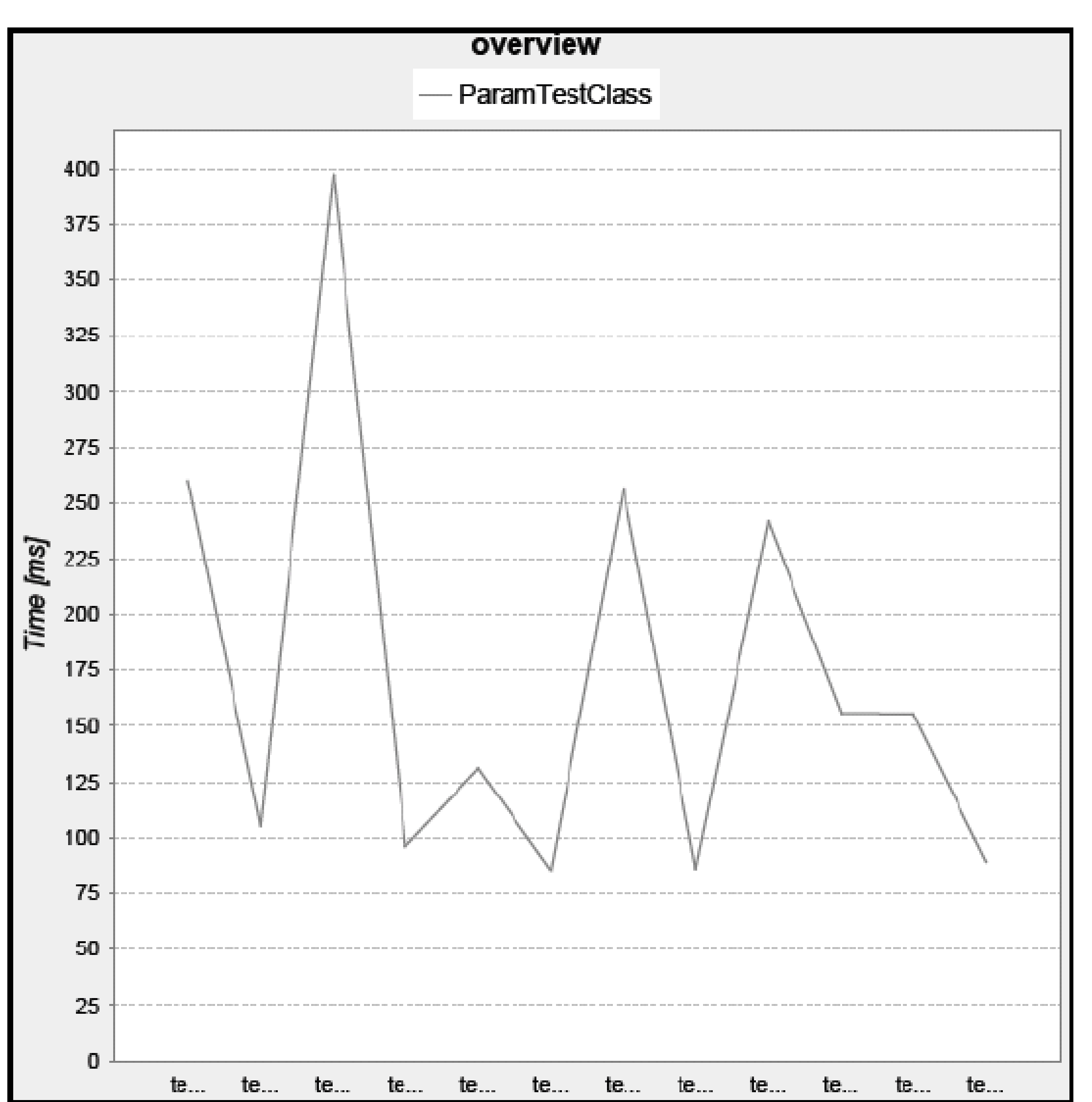


图 10-5　运行结果的图形化显示

测试类一：JUnitTestClass

```
package resultTest;
import org.punit.runner.*;
import param_test.SampleUtil;
import junit.framework.*;
public class JUnitTestClass extends TestCase {
    public static void main(String[] args) {
        new SoloRunner().run(JUnitTestClass.class);
    }
    protected void setUp() throws Exception {
        System.out.println("setUp"); //$NON-NLS-1$
    }
    protected void tearDown() throws Exception {
        System.out.println("tearDown"); //$NON-NLS-1$
    }
    public void testA() {
        SampleUtil.doSomething();
    }
    public void testB() {
        SampleUtil.doSomething();
    }
    public void testC() {
        SampleUtil.doSomething();
    }
}
```

测试类二：DoSomethingTestClass

```
package resultTest;
import org.punit.type.Test;
import param_test.SampleUtil;
public class DoSomethingTestClass implements Test {
    public void setUpBeforeWatchers() throws Exception {
        System.out.println("This setup will not be caculated into the execution time");
        //$NON-NLS-1$
    }
    public void setUpAfterWatchers() throws Exception {
        System.out.println("This setup will be caculated into the execution time");
        //$NON-NLS-1$
    }
    public void tearDownBeforeWatchers() throws Exception {
        System.out.println("This setup will be caculated into the execution time");
        //$NON-NLS-1$
    }
    public void tearDownAfterWatchers() throws Exception {
        System.out.println("This setup will not be caculated into the execution time");
        //$NON-NLS-1$
    }
    public void testA() {
        SampleUtil.doSomething();
    }
    public void testB() {
        SampleUtil.doSomething();
    }
    public void testC() {
        SampleUtil.doSomething();
    }
}
```

测试类三：SimpleTestClass

```
public class SimpleTestClass {
    public void setUp() {
        SampleUtil.doSomething();
    }
    public void tearDown() {
        SampleUtil.doSomething();
    }
    public void testA() {
```

```
            System.out.println("testA");
            SampleUtil.doSomething();
        }
        public void testB() {
            SampleUtil.doSomething();
        }
        public void testC() {
            SampleUtil.doSomething();
        }
}
```

测试类综合及实现：

```
public class AllTestSuite implements TestSuite {
    public Class<?>[] testSuite() {
        return new Class[] {
                JUnitTestClass.class,
                SimpleTestClass.class,
                DoSomethingTestClass.class,
        };
    }
}
```

下面给出测试用例执行及输出结果的具体代码：

```
package resultTest;
import org.punit.reporter.chart.OverviewReporter;
import org.punit.reporter.chart.image.ImageRender;
import org.punit.reporter.chart.pdf.PDFRender;
import org.punit.reporter.stream.file.FileLogger;
import org.punit.runner.Runner;
import org.punit.runner.SoloRunner;
public class ResultTest {
        public static void main(String[] args) {
            Runner runner = new SoloRunner();
            runner.addEventListener(new FileLogger());
            runner.addEventListener(new OverviewReporter(new ImageRender()));
            runner.addEventListener(new OverviewReporter(new PDFRender()));
            runner.run(AllTestSuite.class);
        }
}
```

执行后的测试结果：

```
[solo] Starting resultTest.AllTestSuite
TestSuite: resultTest.AllTestSuite
resultTest.JUnitTestClass
setUp
tearDown
testA() - [402.5168ms]
setUp
tearDown
testB() - [248.657301ms]
setUp
tearDown
testC() - [134.404513ms]
unit_test.SimpleTestClass
testA
testA() - [215.002186ms]
testB() - [449.080819ms]
testC() - [208.451913ms]
resultTest.DoSomethingTestClass
This setup will not be caculated into the execution time
This setup will be caculated into the execution time
This setup will be caculated into the execution time
testA() - [319.857691ms]
This setup will not be caculated into the execution time
This setup will not be caculated into the execution time
This setup will be caculated into the execution time
This setup will be caculated into the execution time
This setup will not be caculated into the execution time
testB() - [165.112047ms]
This setup will not be caculated into the execution time
This setup will be caculated into the execution time
This setup will be caculated into the execution time
This setup will not be caculated into the execution time
testC() - [213.82131ms]
total: 9, failures:0 (GREEN) - 5952.779474ms
```

对测试结果进行分析后发现：三个不同的类执行的时间是不同的，可以用图形的直观结果进行比较，如图 10-6 所示。

对比这三个时间，其实它们执行函数的方法是一样的，但是这三个类的执行时间却不同，因此可以对比出这三个类的性能。

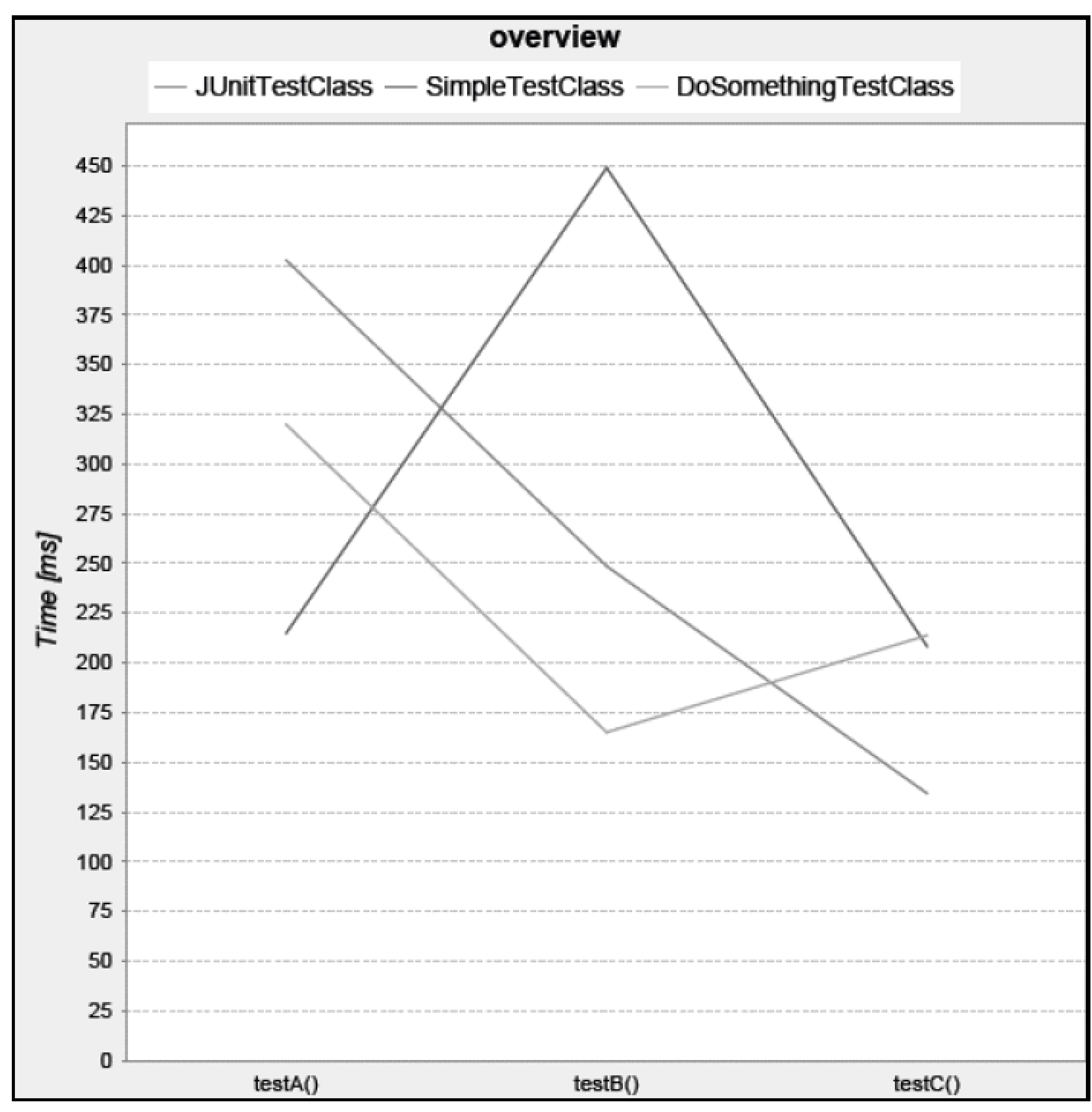


图 10-6　三个不同类的测试用例的执行结果

5. 自动化测试过程

自动化测试售货机程序，结果显示在图 10-7 的右下角。

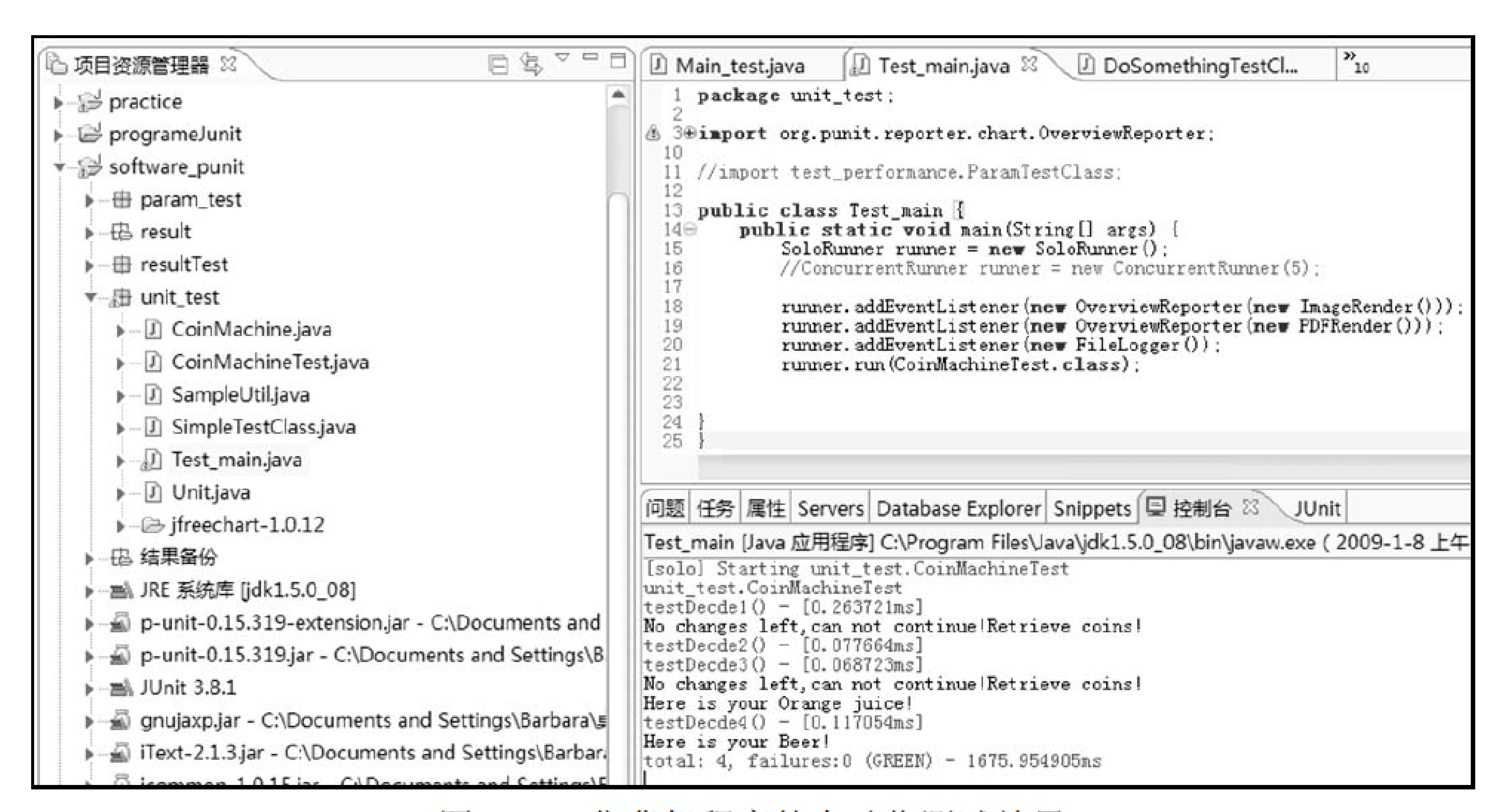


图 10-7　售货机程序的自动化测试结果

p-unit 是一个用于 Java 单元性能测试的工具，由张黄瞩工程师开发，并于 2007 年开始使用，因此网络上的介绍性资源有限。p-unit 这个开源工具使用起来非常方便，还可以对其进行改进。在建立的 Java 项目中加载外部文档后，即可调用 p-unit 包里的东西，而且 p-unit 与 JUnit 是兼容的，因而可以很方便地调用 JUnit 里的类。p-unit 作为一款开源

的性能测试软件，提供了多线程支持、参数化测试用例以及不同虚拟机的性能测试，而且关键是 p-unit 还可以提供可视化的显示结果，除了能够存储文本之外，还可以显示图片及 PDF 文档，更加客观。而且，通过测试几个性能不同的类，可以从结果图中根据曲线的变化看出不同类的性能变化，从而便于选择、改进单元的性能。

当然在使用 p-unit 的过程中，有时会遇到一些困难，如显示图片和 PDF 功能的代码没有作用。但只要经过仔细分析，就会发现是由于没有加载显示图像的 jfreechart 包造成的。这个问题解决后，就可以看到在项目的文件夹中自动产生了名为 result 的文件夹，同时还可以看到产生的图像以及 PDF 格式的可视化结果，由于结果很直观，因此不同类的性能对比也就很明显，从而更便于选择性能良好的类。

总之，p-unit 的出现为单元性能测试带来了极大的便利，人们可以简单地实现不同数量级的一次性性能测试，以及不同环境、不同参数下的单元性能测试，尤其是可以得到可视化的测试结果，这些都将使 Java 单元性能测试的效率大为提高。

实 验 习 题

1. 应用 NTime 工具对.NET 程序进行单元性能测试，给出具体的测试流程和结果，并与 p-unit 进行全面比较。

2. 学习其他的单元性能测试工具(包括商用的)，总结使用这些工具进行单元性能测试时用到的测试技术和测试方法。

第11章　Web应用性能测试工具JMeter

随着 Web 应用的增多，服务器应用解决方案中以 Web 为核心的应用也越来越多，很多公司的各种应用架构都是以 Web 应用为主。一般的 Web 测试和以往的应用程序测试的侧重点不完全相同，在基本功能通过测试后，就要进行重要的系统性能测试了。那么系统的性能是什么呢？系统的性能是一个很大的概念，覆盖面非常广泛，对于一个软件系统而言包括执行效率、资源占用率、稳定性、安全性、兼容性、可靠性等，它可以是功能的开销或者是同步运行功能的数目。特别是当 Web 网站遭遇访问高峰时，容易发生服务器响应速度变慢甚至服务中断等不可接受的极端状况。Web 应用性能测试是为了描述 Web 应用系统与性能相关的特性并对其进行评价，而实施和执行的一类测试。Web 应用性能测试主要检验软件是否达到需求规格说明中规定的各类性能指标，并满足一些性能相关的约束和限制条件。简而言之，Web 性能测试就是模拟大量用户操作给网站造成压力，并评测 Web 系统在不同负载和不同配置下能否达到已经定义的标准，分析和消除与软件结构中相关联的性能瓶颈。

11.1　Web 性能测试工具介绍

Web 应用性能测试包括负载测试和压力测试两个方面。负载测试是为了确定在各种级别负载下系统的性能而进行的测试，其目标是测试当负载逐渐增加时，系统组成部分的相应输出项，如响应时间、连接失败率、CPU 负载、内存使用等如何决定系统的性能。压力测试是为了确定 Web 应用系统的瓶颈或者所能承受的极限性能点而进行的测试，其目标是获得系统所提供的最大服务级别的测试。

系统的负载测试需要采用负载测试工具进行，真实模拟大量用户访问 Web 应用系统来测试系统的表现，包括测试静态 HTML 页面的响应时间，甚至测试动态网页(包括 ASP、PHP、JSP 等)的响应时间等，看是否满足预期的设计指标要求，为服务器的性能优化和调整提供数据依据。

系统的压力测试也需要使用压力测试工具，主要是对 Web 服务器进行压力测试。测试可以帮助找到一些大的问题，如死机、崩损、内存泄露等。因为有些存在内存泄露问

题的程序，在运行一两次时可能不会出现问题，但是如果运行了成千上万次后，内存泄露的越来越多，就会导致系统崩溃。

目前比较流行的负载测试和压力测试工具有 LoadRunner、WebLoad、QALoad、JMeter 等。其中 HP 公司的 LoadRunner 是其中的佼佼者，并已成为行业的规范。

11.1.1　HP LoadRunner

LoadRunner 通过模拟上千万用户实施并发负载及实行实时性能监测的方式来确认和查找问题，能够对整个企业架构进行测试。LoadRunner 适用于各种体系架构，能支持广泛的协议和技术(如 Web、FTP、Database 等)，能预测系统行为并优化系统性能。它通过模拟实际用户的操作行为和实行实时性能监测，来帮助测试人员更快地查找和发现问题。LoadRunner 是一个强大有力的压力测试工具，它的脚本可以录制生成，自动关联；测试场景面向指标，实现了多方监控，而且测试结果采用图表显示，可以自由拆分组合。

通过对 LoadRunner 的测试结果图表进行对比，可以寻找出造成系统瓶颈的原因。一般来说，可以按照服务器硬件、网络、应用程序、操作系统、中间件的顺序进行分析。

LoadRunner 是一款收费软件，根据测试项目和虚拟用户数目的不同而需支付不同的费用。

11.1.2　Apache JMeter

Apache JMeter 是一个专门为进行服务器负载测试而设计的、100%的 Java 桌面应用程序。用于对 C/S 结构的软件(例如 Web 应用程序)进行负载测试和压力测试。原先它是为 Web/HTTP 测试而设计的，但是它已经扩展到支持各种各样的测试模块。JMeter 可以用于测试静态和动态资源，例如静态文件、Java 小服务程序、CGI 脚本、Java 对象、数据库、FTP 服务器等。JMeter 可以用于在一个服务器、网络或者对象上模拟重负载，来测试它的强度或者分析在不同负载类型下的全面性能。

另外，JMeter 支持对应用程序进行功能/回归测试，通过创建带有断言的脚本来验证被测程序所返回的结果与期望结果是否一致。为了保持最大限度的灵活性，JMeter 允许使用正则表达式创建断言。

在设计阶段，JMeter 能够充当 HTTP Proxy(代理)来记录 IE/Netscape 的 HTTP 请求，也可以记录 Apache 等 Web Server 的 log 文件来重现 HTTP 流量。当这些 HTTP 客户端请求被记录以后，测试运行时便可以方便地设置重复次数和并发度(线程数)来产生巨大的流量。JMeter 还提供可视化组件和报表工具，用于将服务器在不同压力下的性能展现出来。

相比其他 HTTP 测试工具，JMeter 最主要的特点在于扩展性强。JMeter 能够自动扫描其 lib/ext 子目录下.jar 文件中的插件，并将其装载到内存，让用户通过不同的菜单来调用。

11.2　应用 JMeter 进行 Web 性能测试

11.2.1　JMeter 测试环境建立

搭建 JMeter 测试环境的过程非常简单，由于 JMeter 是一个图形界面配置工具，并且不像 LoadRunner 那样定位为高端测试人员专用，所以 JMeter 能够让普通 Web 应用开发人员快速上手。

1. 软件下载

搭建 JMeter 测试环境所需要下载的工具均是开源软件。

JDK1.5

http://java.sun.com/javase/downloads/index.jsp

Apache tomcat 5.5

http://tomcat.apache.org/download-55.cgi

jakarta-jmeter-2.2

http://jakarta.apache.org/site/downloads/downloads_jmeter.cgi

sqljdbc_1.1.1501.101_enu.exe

http://www.microsoft.com/downloads/details.aspx?FamilyId=6D483869-816A-44CB-9787-A866235EFC7C&displaylang=en

2. 搭建 JMeter 测试环境

(1) 安装 JDK (1.4 以上)。

下载 jdk-1_5_0_09-nb-5_0-win-ml.exe，直接双击以默认方式安装，一般安装至 C:\Program Files\Java 目录下。

设置环境变量：右击“我的电脑”，执行“高级”|“环境变量”|“系统变量”。在系统变量框里进行环境变量设置。

① 新建变量 JAVA_HOME，值为：安装 JDK 的目录。

② 新建变量 CLASSPATH，值为：%JAVA_HOME%\lib; %JAVA_HOME%\bin。

③ 在 path 变量后加上：%JAVA_HOME%\lib;%JAVA_HOME%\bin。

在命令提示符窗口中输入“JAVA”或“JAVAC”，如果出现帮助信息，则 JDK 安装成功。

(2) 安装 ApacheTomcat。

下载 apache-tomcat-5.5.20.exe(或更高版本)，直接双击按照默认路径安装，一般安装至 C:\Program Files\Apache Software Foundation\Tomcat 5.5 目录下。然后在 Tomcat 中部署简单的 Web 应用，这个应用是待测试的案例，现实使用中这可能是一个已经存在的 Web

应用系统，也可能是一个正处于构建中的 Web 应用系统。11.1.2 节将详细说明这个 Web 应用案例。

(3) 安装 JMeter。

解压 jakarta-jmeter-2.2.zip 文件至 C:\jakarta-jmeter-2.2 目录下（当前的 JMeter 最新版本是 2.6）。

在桌面上右击“我的电脑”，执行“高级”|“环境变量”，然后执行“系统变量”|“新建”。在变量名中输入“JMETER_HOME”，在变量值中输入“C:\jakarta-jmeter-2.2”，在 CLASSPATH 变量值中添加%JMETER_HOME%\lib\ext\ApacheJMeter_core.jar、%JMETER_HOME%\lib\jorphan.jar 和%JMETER_HOME%\lib\logkit-1.2.jar，然后确定即可。

JMeter 有如下几个目录：bin、docs、extras、lib 和 printable_docs。进入 bin 目录，运行下面的 jmeter.bat 就可以看见 JMeter 的 GUI 客户端了，可以对测试进行相关的配置。如果要对自己开发的网站进行测试，则需要运行网站应用服务器，然后再将访问地址录入到 JMeter 中进行测试。

lib 目录下有两个目录，一个是 ext 目录，另一个是 junit 目录。ext 目录用于存放用户对 JMeter 进行扩展的测试应用，而用户的扩展所依赖的包则要直接放在 lib 目录下(而不是 lib/ext 下)。JMeter 会自动从它的/lib 和/lib/ext 目录中的 jar 包中发现类。如果用户开发了新的 JMeter 组件，则可将它们以 jar 包形式复制到 JMeter 的/lib/ext 目录下。JMeter 将会自动发现在这里的任何 jar 包的 JMeter 组件。

(4) JMeter 针对不同的 Web 应用测试可能还需要 SSL 加密、JDBC 驱动、Apache SOAP 以及 BeanShell 等包。

① SSL 加密：为了测试一个使用 SSL 加密(HPPS)的 Web 服务器，JMeter 需要提供一个 SSL 实现(例如 Sun 的 Java Secure Sockets Extension(JSSE))。包含需要的加密包到 JMeter 的 classpath。同时，通过注册 SSL 提供者更新 system.properties 文件。

② JDBC 驱动：如果需要做 JDBC 测试，则需要添加厂商的 JDBC 驱动到 classpath。确认文件是一个 jar 文件，而不是 zip 文件。

③ Apache SOAP：Apache SOAP 需要 mail.jar 和 activation.jar。这时需要下载并复制这两个 jar 文件到 jmeter/lib 目录下。一旦文件放到那里，JMeter 就会自动找到它们。

④ BeanShell：为了运行 BeanShell 函数或任何 BeanShell 测试元件(取样器、定时器等)，需要从 http://www.beanshell.org/下载 BeanShell 的 jar 文件并将其复制到 jmeter/lib 目录下，JMeter 会自动找到它。

11.2.2　JMeter 测试功能及使用流程

1. JMeter 的主要功能

JMeter 的主要功能包括：

(1) 能够对 HTTP 和 FTP 服务器进行压力测试和性能测试， 也可以对任何数据库进行同样的测试(通过 JDBC)。

(2) 完全的可移植性和 100％纯 Java。

(3) 完全 Swing 和轻量组件支持包(预编译的 JAR 使用 javax.swing.*)。

(4) 完全多线程。框架允许通过多个线程并发取样，以及通过单独的线程组对不同的功能同时取样。

(5) 精心的 GUI 设计允许快速操作和更精确的计时。

(6) 缓存和离线分析/回放测试结果。

JMeter 同时还具有高可扩展性：

(1) 可链接的取样器允许无限制的测试能力。

(2) 各种负载统计表和可链接的计时器可供选择。

(3) 数据分析和可视化插件提供了很好的可扩展性以及个性化。

(4) 具有提供动态输入到测试的功能(包括 JavaScript)。

(5) 支持脚本编程的取样器(在 1.9.2 及以上版本中支持 BeanShell)。

2. JMeter 的主要元件

JMeter 的主要元件有：

(1) 测试计划是使用 JMeter 进行测试的起点，是其他 JMeter 测试元件的容器。

(2) 线程组代表一定数量的并发用户，可以用来模拟并发用户发送请求。实际的请求内容在 Sampler 中定义，它被线程组包含。

(3) 监听器负责收集测试结果，同时也被告知了结果显示的方式。

(4) 逻辑控制器可以自定义 JMeter 发送请求的行为逻辑，它与 Sampler 结合使用可以模拟复杂的请求序列。

(5) 断言可以用来判断请求响应的结果是否如用户所期望的。它可以被用来隔离问题域，即在确保功能正确的前提下执行压力测试。这个限制对于有效的测试是非常有用的。

(6) 配置元件负责维护 Sampler 所需要的配置信息，并根据实际的需要修改请求的内容。

(7) 前置处理器和后置处理器负责在生成请求之前和之后完成工作。前置处理器常常用来修改请求的设置，后置处理器则常常用来处理响应的数据。

(8) 定时器负责定义请求之间的延迟间隔。

3. JMeter 的测试流程

在测试环境搭建好之后，就可以开始进行测试了。使用 JMeter 进行测试主要包含以下几个步骤，当然最初 JMeter 必须是运行着的，如图 11-1 所示(在 JMeter 的 bin 目录下找到 jmeter.bat 批处理文件，双击打开)。

1) 建立测试计划

测试计划描述了测试执行过程中 JMeter 的执行过程和步骤，一个完整的测试计划包括一个或多个线程组(Thread Groups)、逻辑控制器(Logic Controller)、样例产生控制器(Sample Generating Controllers)、监听器(Listener)、定时器(Timer)、断言(Assertions)、配置元素(Config Elements)。启动 JMeter 时，它已经建立了一个默认的测试计划。一个 JMeter

应用实例只能建立或者打开一个测试计划。



图 11-1　启动 JMeter

2) 添加线程组

线程组是一个测试计划中最先建立的部分。在线程组中，测试者将要指定同时将有多少个线程并发访问；每两个线程访问之间的间隔时间(Ramp-Up Period)；总共要循环访问多少次，可以选择具体数字也可以选择无限期循环，如图 11-2 所示。

3) 添加取样器

请求是 Web 测试的主体，通过向指定服务器提交特定请求并捕捉返回状态，来检测 Web 应用的性能。JMeter 中可添加的特定请求由取样器来选择和发送，其中包括 HTTP



图 11-2　线程组的选择及配置

图 11-2　(续)

请求、FTP 请求、JDBC 数据库访问请求、LDAP 请求、AJP1.3 请求、SOAP 请求、JMS 请求等，如图 11-3 所示。本章中主要展示 HTTP 请求的测试过程。

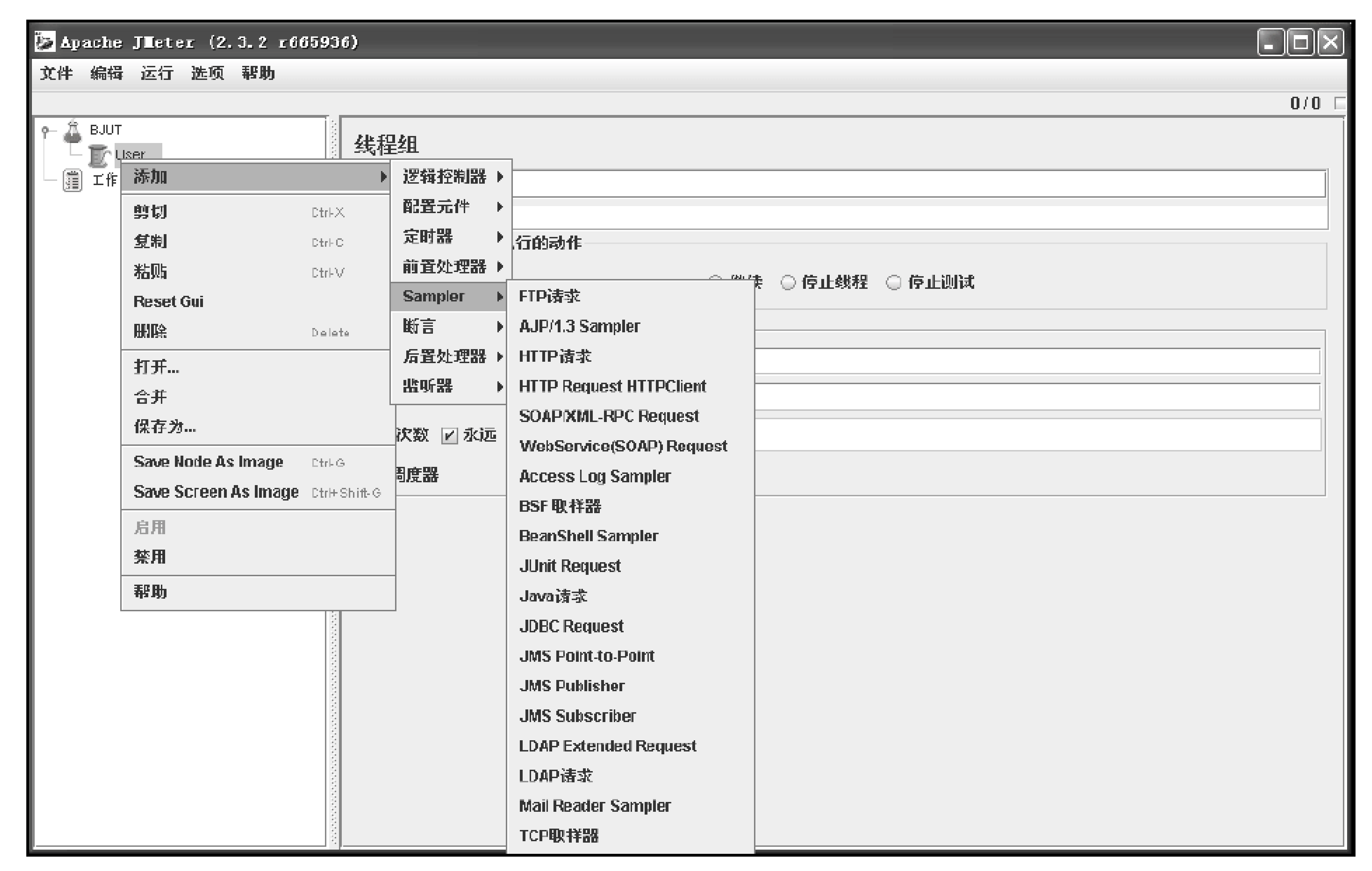


图 11-3　取样器的可选内容

每个取样器都有一些可以设置的属性。可以通过添加一个或多个配置元件到取样器来进一步定制它。注意 JMeter 发送请求是按照取样器出现在树中的顺序来进行的。

如果想发送多个类型相同的请求(例如 HTTP 请求)到相同的服务器，可以考虑使用一个默认配置元件。每个控制器都有一个或多个默认配置元件。

记着添加一个监听器到线程组来查看或保存请求结果至磁盘。

4) 添加监听器

监听器为了记录测试信息及使用 JMeter 提供的可视化界面查看测试结果，提供了许多结果分析方式以供选择，测试者可以根据自己习惯的分析方式来选择不同的结果显示方式，如图 11-4 所示。

以上几步是进行 JMeter 性能测试的必要步骤，JMeter 同时还存在一些可选步骤。

图 11-4　监听器的可选内容

添加逻辑控制器：逻辑控制器允许测试人员定制当发送请求时 JMeter 使用的判断逻辑，如图 11-5 所示。逻辑控制器还可以作为下列任何元件的子元件：取样器(请求)、配置元件及其他逻辑控制器。逻辑控制器可以改变来自它们的子元件的请求顺序。它们还可以修改请求本身，从而导致 JMeter 重复请求。



图 11-5　逻辑控制器的可选内容

配置元组：实际的测试工作往往是针对同一个服务器上的 Web 应用展开的，所以 JMeter 提供了这样一种设置，就是将默认请求属性设置成被测试服务器的相关属性，如

图 11-6 所示。以后的同等请求设置中就可以忽略这些相同参数的设置，以减少设置参数录入的时间。

添加断言：断言允许测试人员给出关于从测试服务器接收到的响应的行为，如图 11-7 所示。使用断言可以测试应用程序是否返回期望的结果。

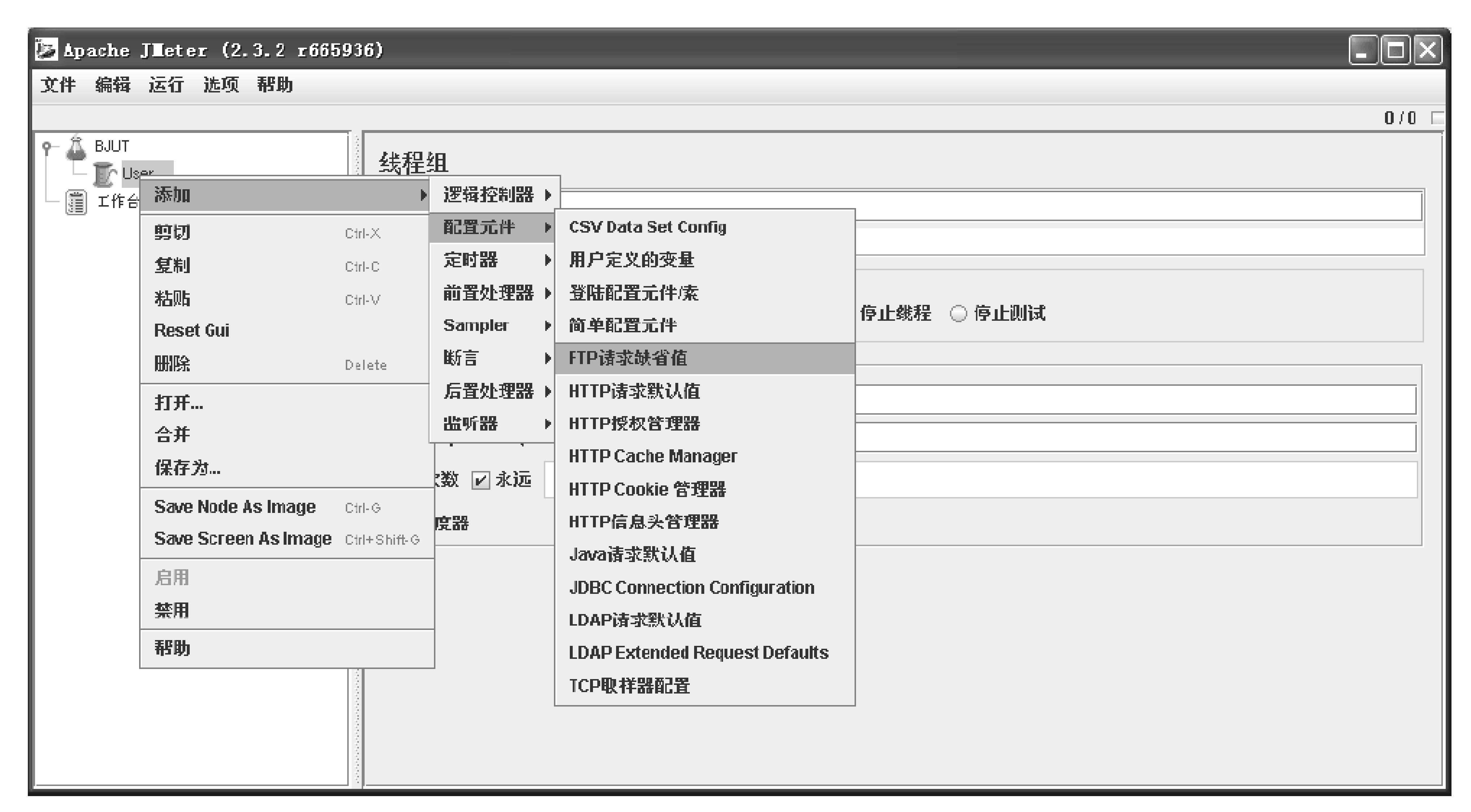


图 11-6　可配置的请求默认值



图 11-7　断言的相关内容

添加定时器：定时器可以让测试在指定的时间进行，也可以在指定时间周期之后才开始运行，如图 11-8 所示。默认情况下，JMeter 线程发送请求时不请求暂停。建议通过添加一个可用的定时器到线程组来指定一个延迟。如果不添加延迟，JMeter 会在短时间内产生太多请求，这可能会压垮测试中的服务。

定时器会使 JMeter 在一个线程开始每个请求时延迟一段时间。如果选择添加多于一个定时器到一个线程组，JMeter 会在执行取样器前获得定时器数量并暂停那个时间量。

添加前置处理器和后置处理器：前置处理器和后置处理器可以在请求的开始之前和结束之后分别对请求做一些特定的操作，比如 URL 重写，如图 11-9 所示。

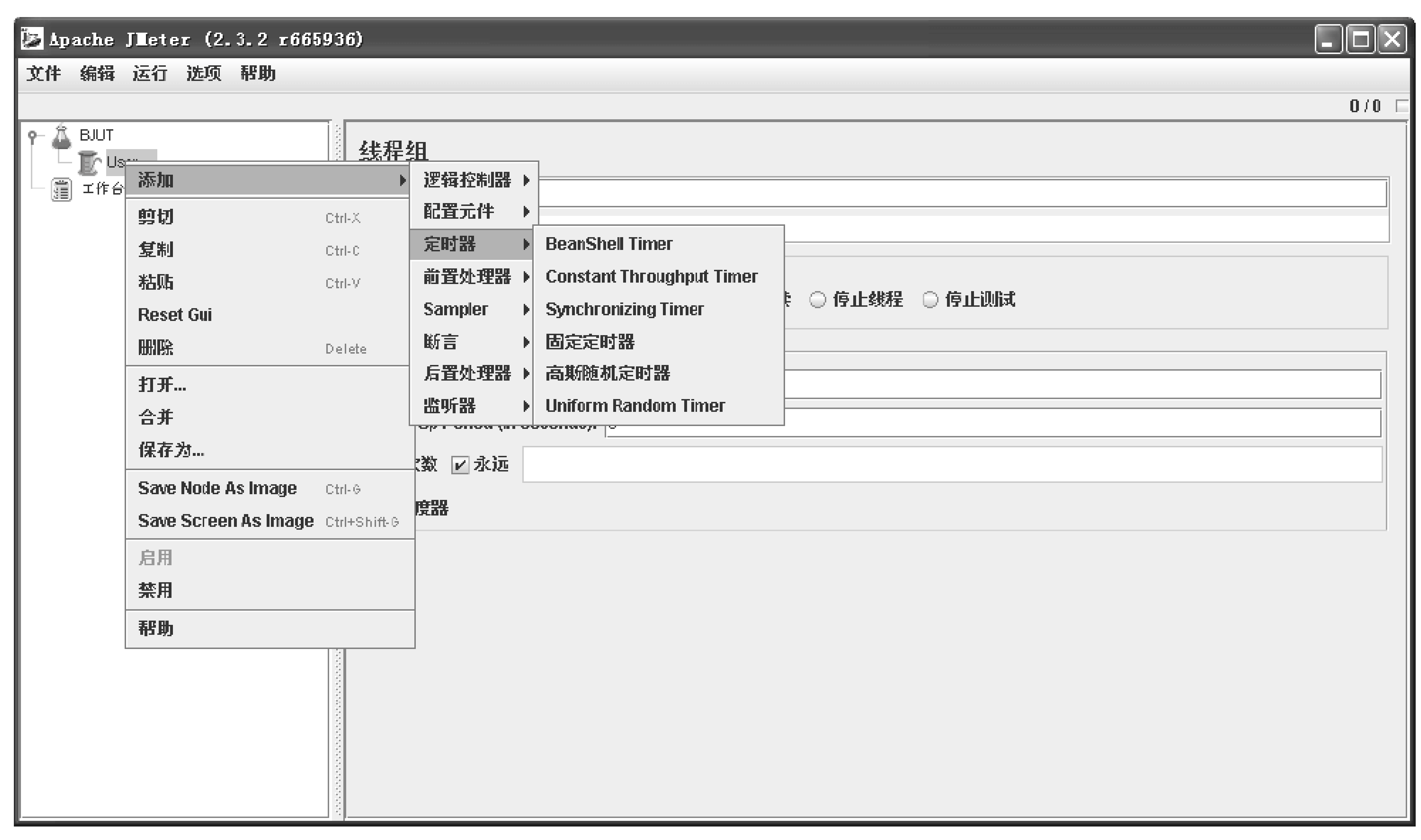


图 11-8　定时器的可选内容





图 11-9　前置处理器和后置处理器的可选内容

JMeter 的执行顺序是：定时器、取样器、后置处理器(如果 SampleResult 不为空)、断言(如果 SampleResult 不为空)、监听器(如果 SampleResult 不为空)。

11.3　JMeter 测试应用举例

下面针对 JMeter 的几种典型测试，分别给出相应的例子。

11.3.1　测试 HTTP 请求

创建 5 个用户，分别向 Jackrta 网站上的两个网页发送请求。每个用户运行测试两次。这样，总的 HTTP 发送请求次数为 20(5 个用户×2 次请求×重复 2 次)。

1. 添加用户

处理 JMeter 测试计划的第一步就是添加线程组元件。这个线程组会模拟用户数量，用户应该发送请求的频率以及应该发送的数量。

下面来添加一个线程组：首先选择这个测试计划，用鼠标右击它，然后在弹出的菜单中选择“添加”|“线程组”命令：首先为这个线程组起一个有意义的名字，在 Name 文本框中，输入 Jakarta Users；下一步，增加用户的数量(线程)为 5；在 Ramp-Up Period 文本框中，使用默认值 0，该属性表示每个用户启动的迟延时间；最后，禁用 Forever 复选框并设置循环次数为 2，该属性表示测试的重复次数，如图 11-10 所示。

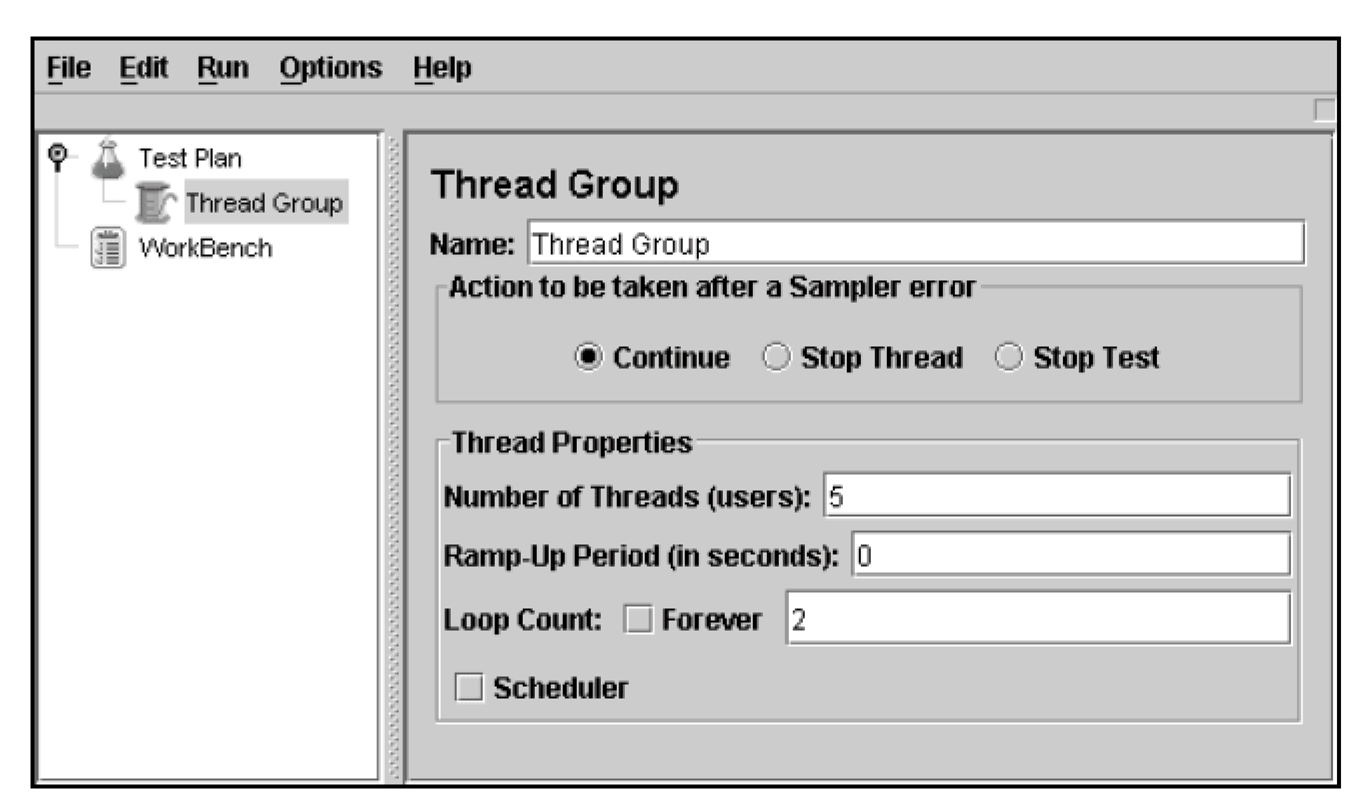


图 11-10　线程组参数设置

2. 添加 HTTP 请求默认属性

首先选择 Jakarta Users 元件，用鼠标右击并在弹出的菜单中选择“添加”|“配置元件”|“HTTP 请求默认值”命令(HTTP 请求默认值元件并不告诉 JMeter 来发送 HTTP 请求，它仅定义这个 HTTP 请求所使用的默认值)。然后选择这个新元件，以显示其控制面板。

对于创建的测试计划，所有的 HTTP 请求都将发送到相同的 Web 服务器 jakarta.apache.org，如图 11-11 所示。

3. 添加 Cookies 支持

除非应用程序明确不使用 Cookies，否则几乎所有的网站应用程序都会使用 Cookies 支持。要添加 Cookies 支持，可以简单地在测试计划中给每一个线程组添加一个 HTTP Cookie 管理器，以确保每个线程组都有自己的 Cookies，并能共享跨越所有的 HTTP 请求对象。

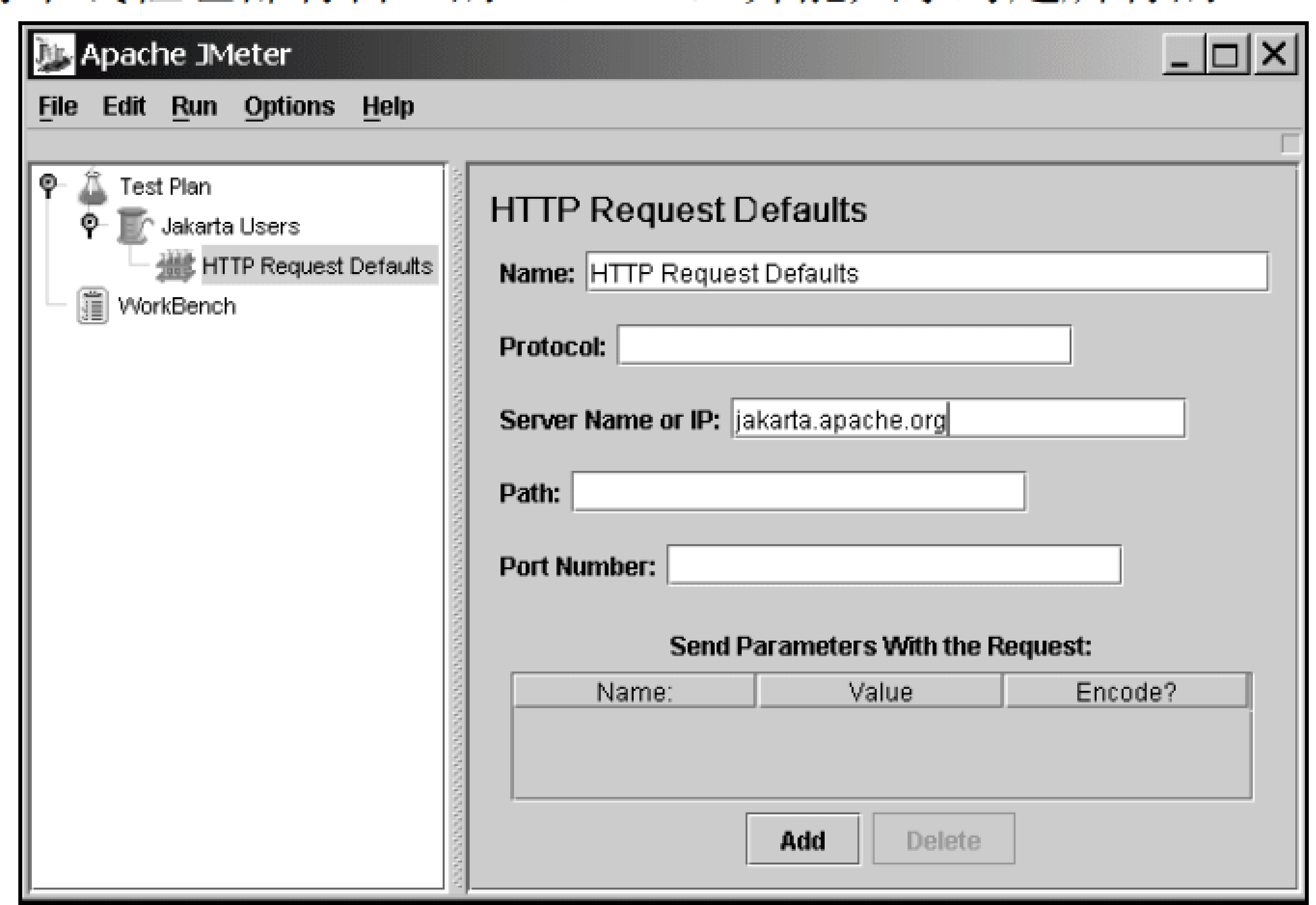


图 11-11　设置 HTTP 请求属性

要添加 HTTP Cookie 管理器，可选择这个线程组，选择“添加”|“配置元件”|“HTTP Cookie 管理器”，也可以从编辑菜单或通过鼠标右击来实现添加。

4. 添加 HTTP 请求

在这个测试计划中需要实现两个 HTTP 请求：一个是 Jakarta 网站首页(http://jakarta.apache.org/)，另一个是工程向导网页(http://jakarta.apache.org/site/guidelines.html)。JMeter 将按照它们在树中出现的次序来发送请求。

首先来为 Jakarta Users 元件添加第一个 HTTP 请求(“添加”|“取样器”|“HTTP 请求”)。然后从树中选择 HTTP 请求元件并修改相关属性：更改名称为 Home Page；设置路径为“/”。不必设置服务器名，因为已在 HTTP 默认请求元件中指定了这个值。然后添加第二个 HTTP 请求并修改相关属性：更改名称为 Project Guidelines；设置路径为 /site/guidelines.html，如图 11-12 所示。

5. 添加一个监听器来浏览/存储测试结果

监听器用于将所有的 HTTP 请求结果存储在一个文件中并显现出数据的可视模型。选择 Jakarta Users 性能测试元件，然后添加一个“图形结果”监听器(“添加”|“监听器”|“图形结果”)。接着，指定一个文件路径和输出文件名，如图 11-13 所示。

标题解释：No of Samples(样本数目)是总共发送到服务器的请求数；Latest Sample(最新样本)是代表时间的数字，是服务器响应最后一个请求的时间；Throughput(吞吐量)是服务器每分钟处理的请求数；Average(平均值)是总运行时间除以发送到服务器的请求数；Median(中间值)是代表时间的数字，有一半的服务器响应时间低于该值而另一半高于该值；Deviation(偏离)表示服务器响应时间变化、离散程度测量值的大小，或者换句话说就是数据的分布。

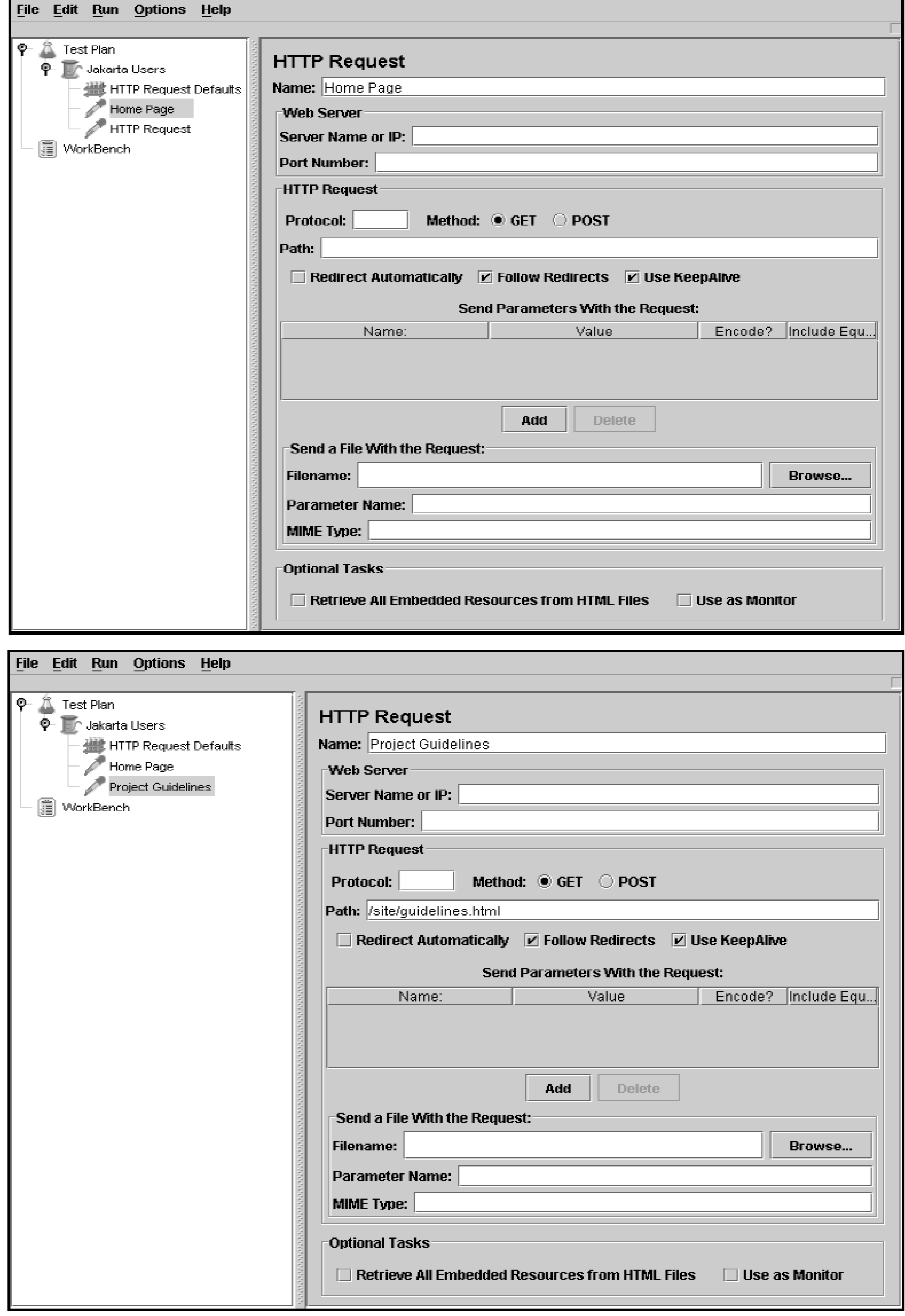


图 11-12　设置 HTTP 请求

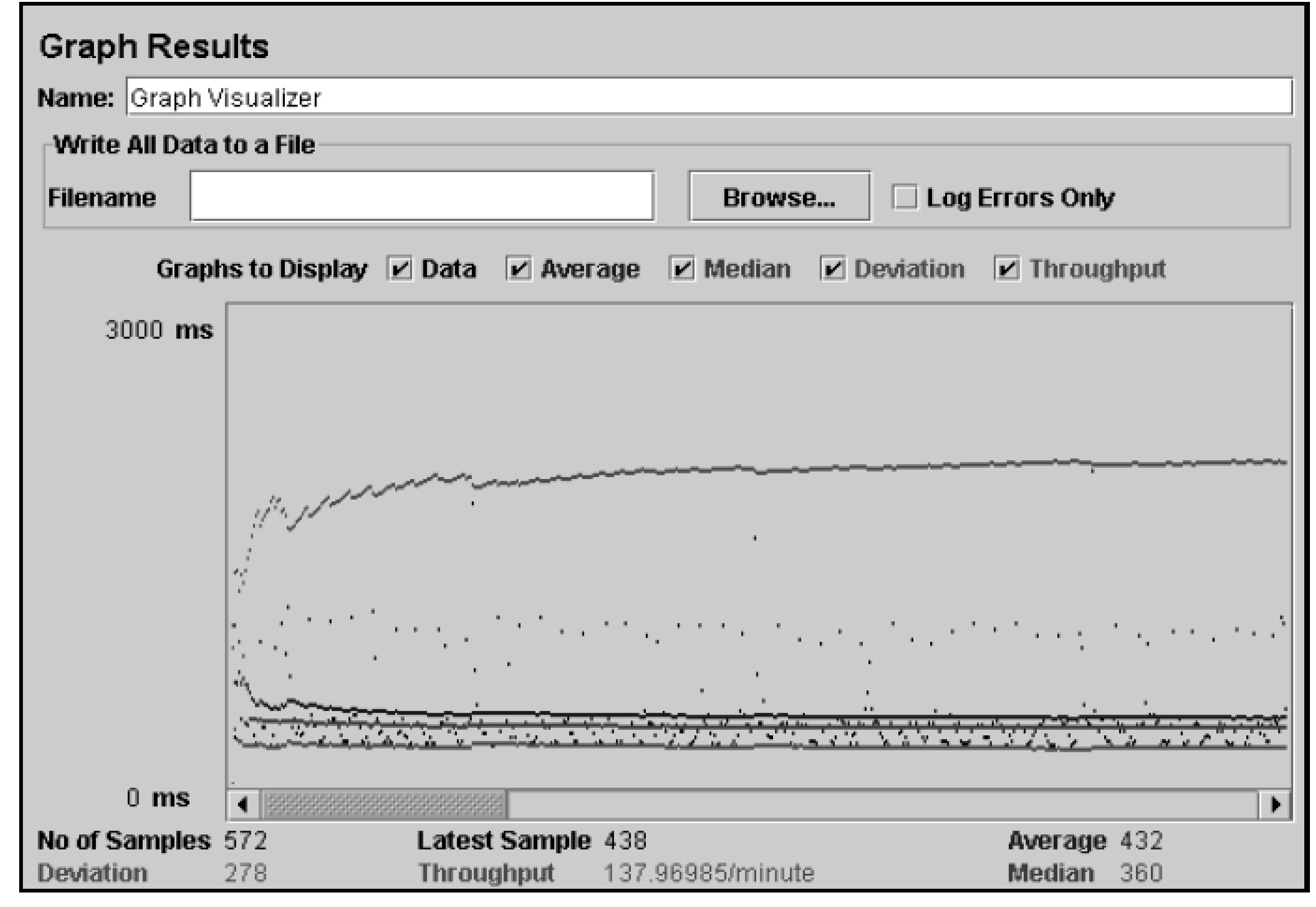


图 11-13　HTTP 请求的监听图形结果

11.3.2　FTP 测试

为 Apache 的 FTP 站点上的两个文件创建 4 个发送请求的用户，每个用户运行两次，所以整个测试有(4 个用户)*(2 个请求)*(重复 2 次)=16 个 FTP 请求，如图 11-14(a)所示。

这里选择进行测试的 FTP 是 Apache 在日本的镜像站点 ftp://ftp.ring.gr.jp，如图 11-14(b)所示。

(1) 添加用户和添加默认 FTP 请求配置基本上与 HTTP 测试的操作相同。

(2) 添加 FTP 请求。

在测试计划中需要制作两个 FTP 请求。一个是 CHANGES 文件（ftp://ftp.ring.gr.jp/pub/net/apache/httpd/CHANGES_2.0），另一个是 HEADER 文件（ftp://ftp.ring.gr.jp/pub/net/apache/httpd/HEADER.html）。同样，JMeter 按照它们在树中出现的顺序来发送请求。

首先添加第一个 FTP 请求到 Apache FTP 元件(“添加” | “取样器” | “FTP 请求”)，然后在树中选择 FTP 请求元件，并编辑以下属性：修改名称为“CHANGES”；修改 Remote File 文本框为“pub/net/apache/httpd/CHANGES_2.0"，修改用户名为“anonymous”；修改密码为“anonymous "，如图 11-14(c)所示。

然后添加第二个 FTP 请求，并修改以下几个属性：修改名称为“Header.html”，修改 Remote File 文本框为“pub/net/apache/httpd/HEADER.html”，修改用户名为"anonymous"，修改密码为"anonymous"，如图 11-14(d)所示。

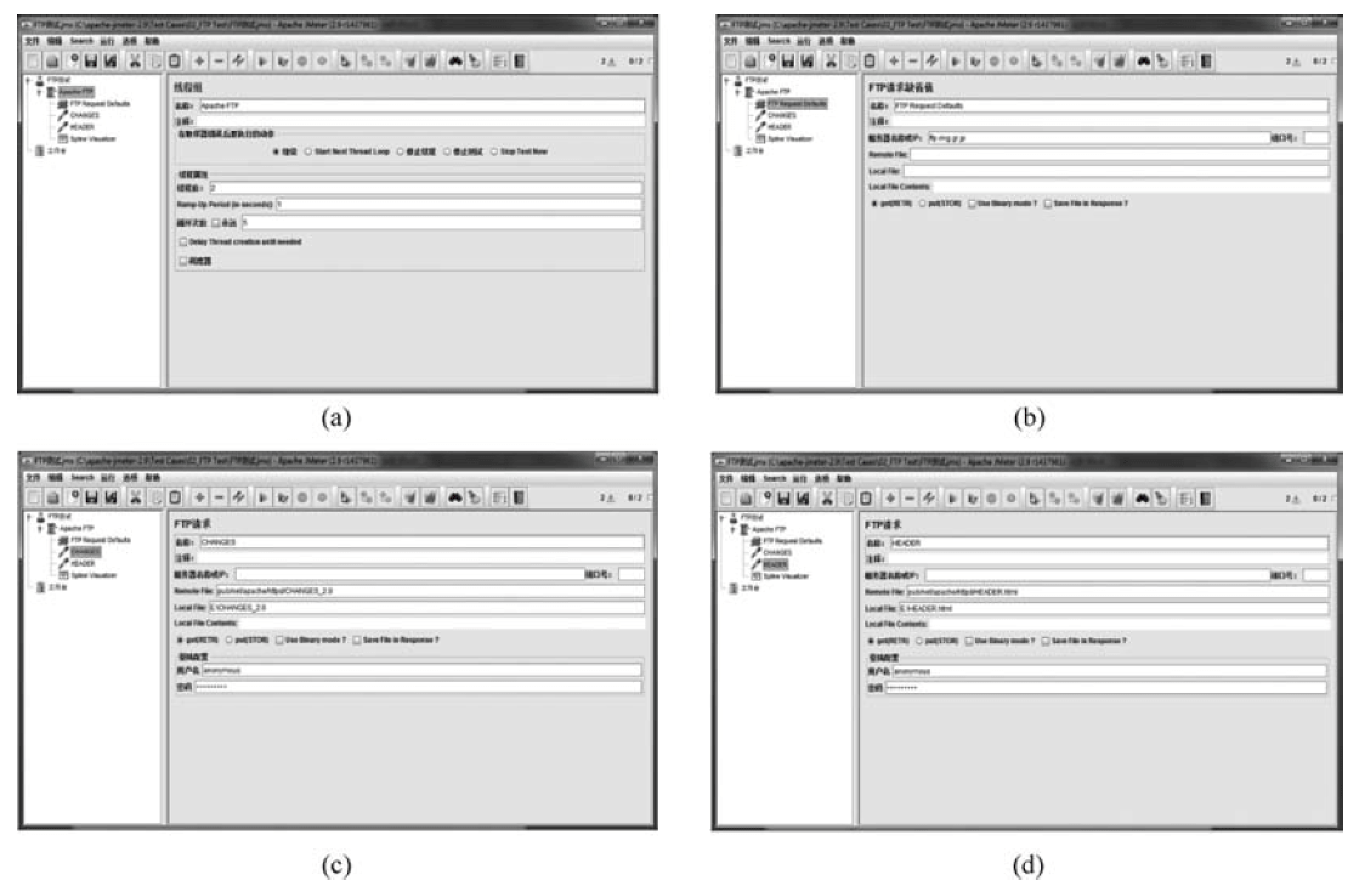

(a)　(b)　(c)　(d)

图 11-14　FTP 请求设置

(3) 添加一个监听器来浏览/存储测试结果。

添加到测试计划的最后元件是一个监听器。这个元件负责存储所有的 FTP 请求结果到文件中，并展示一个可视化的数据模型。

选择 Apache FTP 元件，添加一个 Spline Visualizer 监听器，如图 11-15 所示。

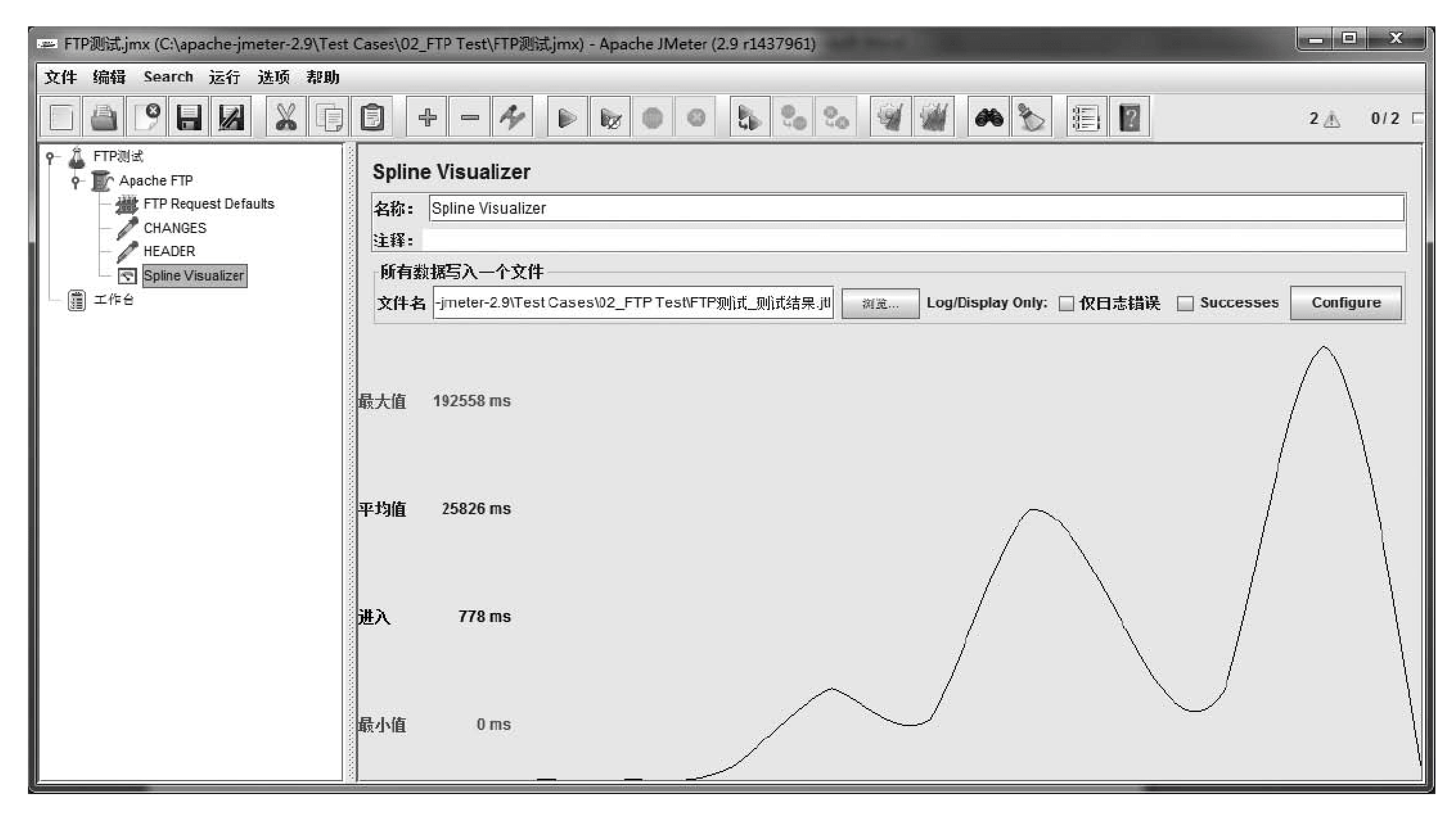


图 11-15　FTP 请求监听曲线结果

11.3.3　数据库测试

创建 10 个用户来向数据库服务器发送两次 SQL 请求，总的 JDBC 请求数量就是(10 用户)×(两次请求)×(重复三次)=60。

这里使用了 MySQL 数据库驱动。要使用这个驱动，它所包含的.jar 文件就必须复制到%JMETER_HOME%\lib 目录下。

(1) 添加用户和添加默认 JDBC 请求配置基本上与 HTTP 测试的操作相同。

(2) 添加 JDBC 请求。

首先选择 JDBC 用户元件，用鼠标右击它，在弹出的菜单中选择“添加”|“配置元件”|“JDBC 连接配置”命令。接着，选择这个新元件以显示它的控制面板，如图 11-16 所示。

设置以下文本框(这里假定用一个名为 test 的本地 MySQL 数据库)：

① 绑定到池变量：这需要能够唯一标识这个配置，用于 JDBC 取样器的识别。

② 数据库 URL：jdbc: mysql://localhost:3306/mydb。

③ JDBC 驱动类：com.mysql.jdbc.Driver。

④ 用户名：guest。

⑤ 密码：gassword。

再次选择 JDBC 用户元件，用鼠标右击它，在弹出的菜单中选择“添加”|Sampler|“JDBC 请求”命令。

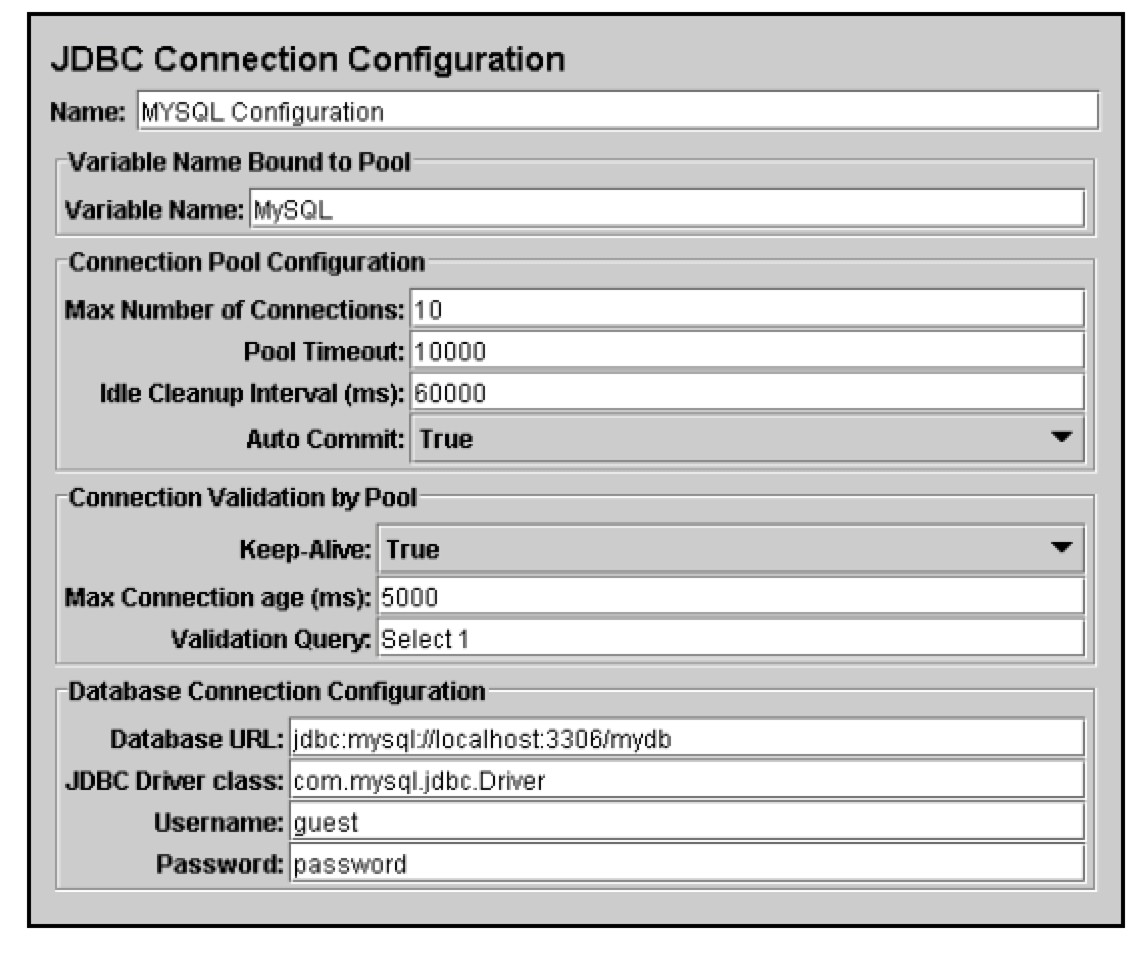


图 11-16　JDBC 请求设置

在这个测试计划中，将发送两个 JDBC 请求。第一个是向 Eastman Kodak stock，第二个是向 Pfizer stock。同样，JMeter 按照它们在树中出现的顺序来发送请求。

(3) 编辑相关属性。

第一个属性：修改名称为“Kodak”，输入变量名“MySQL”(在配置元件里面一样)，输入 SQL 查询字符串。第二个属性：修改名称为“Pfizer”；输入 SQL 查询字符串，如图 11-17 所示。

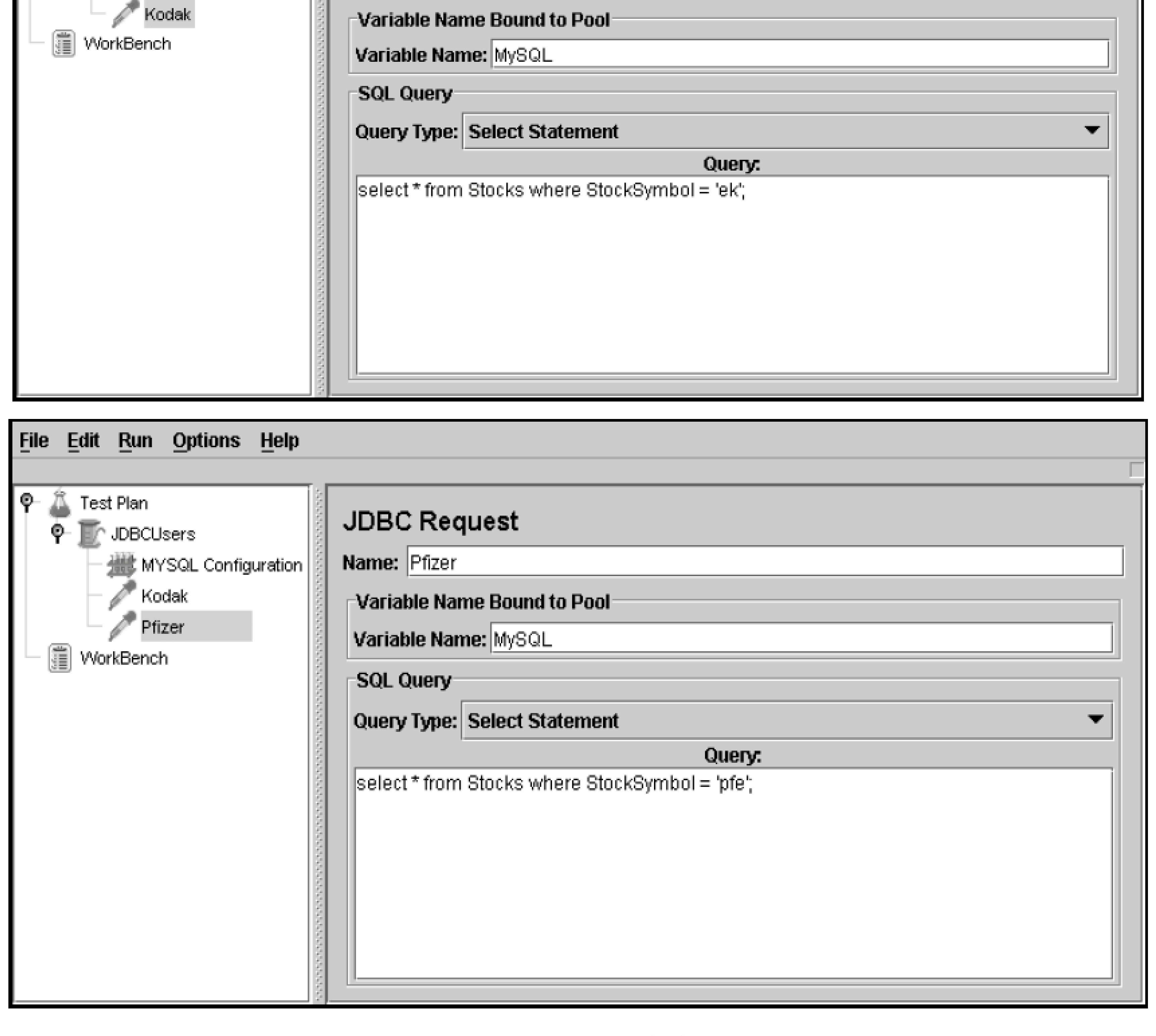


图 11-17　JDBC 请求设置

选择 JDBC 用户元件，添加一个图形结果监听器(“添加”|“监听器”|“图形结果”)，如图 11-18 所示。

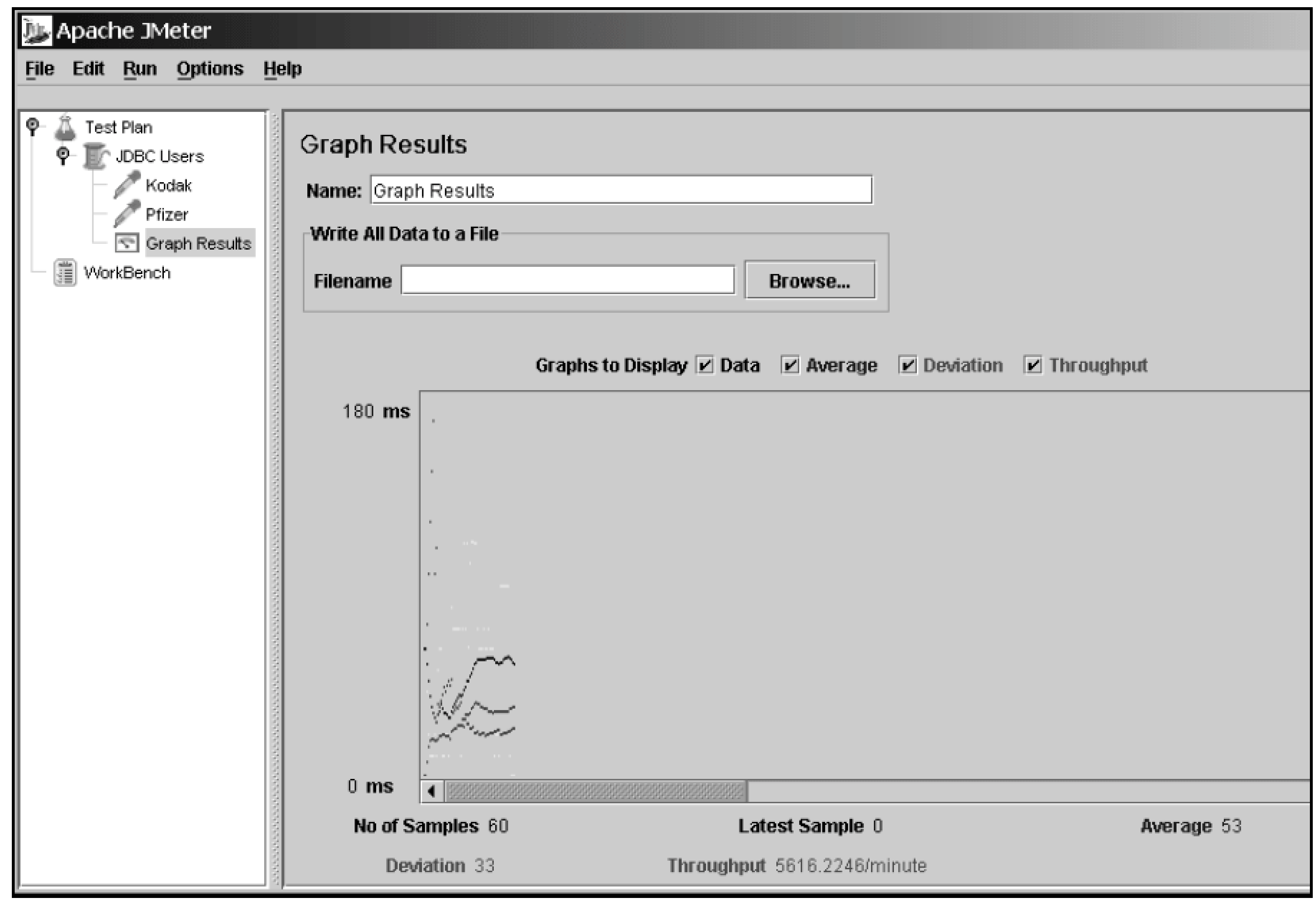


图 11-18　JDBC 图像监听结果

11.3.4　Web 应用测试

过程与 HTTP 测试基本相同，下面用不同的方式来查看性能测试结果(各种显示结果的含义可到 JMeter 官方网站阅读相关手册查看)。

(1) 附带断言结果与图形结果如图 11-19 所示。



图 11-19　基于断言监听的图形结果

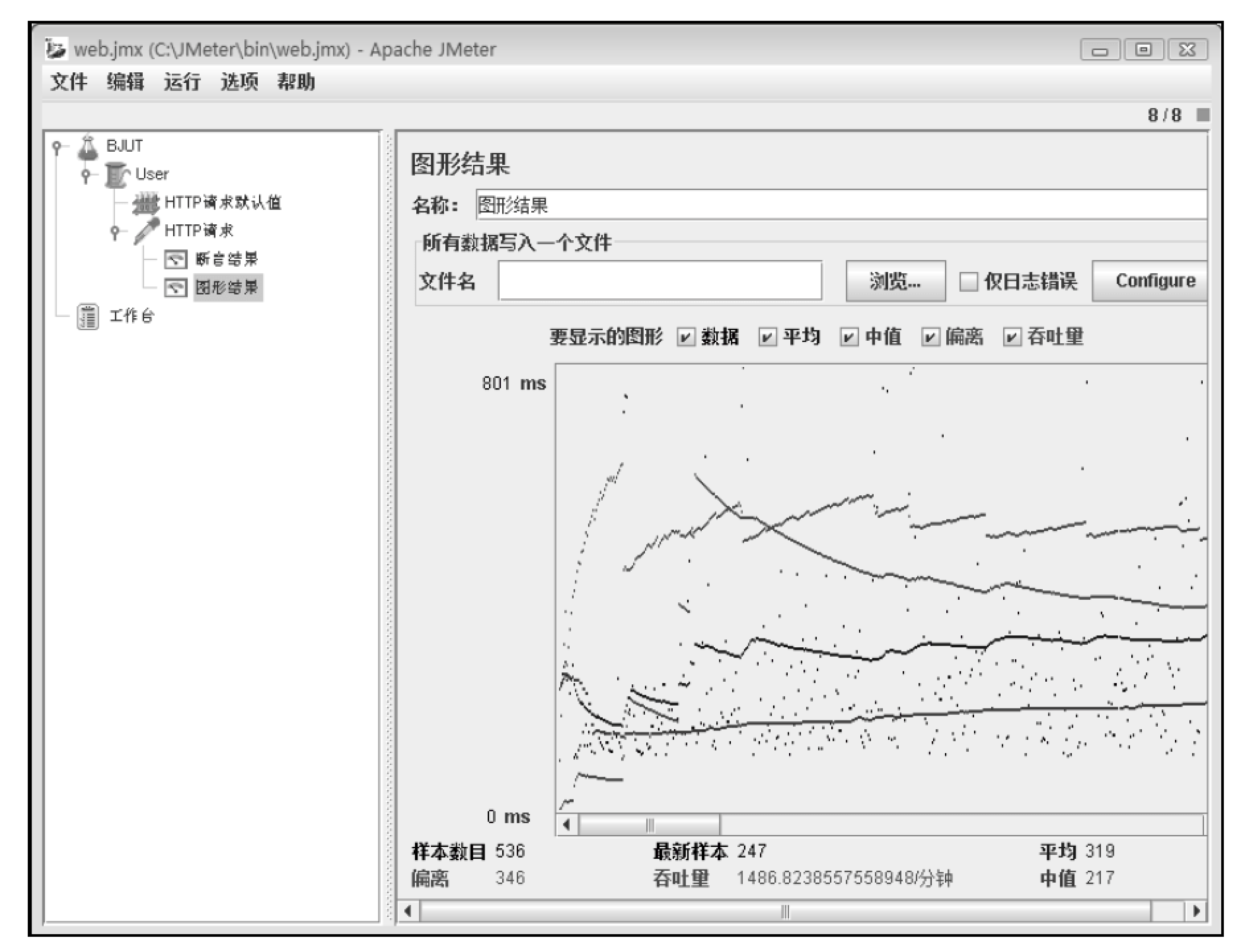


图 11-19 （续）

(2) 用表格查看结果及结果树如图 11-20 与图 11-21 所示。



Sample #	Start Time	Thread Name	Label	Sample Tim...	成功?	Bytes
1	03:54:20.870	User 1-5	HTTP请求	246	✔	5963
2	03:54:20.860	User 1-1	HTTP请求	273	✔	5963
3	03:54:20.872	User 1-7	HTTP请求	276	✔	5963
4	03:54:20.867	User 1-3	HTTP请求	300	✔	5963
5	03:54:20.868	User 1-4	HTTP请求	312	✔	5963
6	03:54:20.873	User 1-8	HTTP请求	320	✔	5963
7	03:54:20.871	User 1-6	HTTP请求	334	✔	5963
8	03:54:20.865	User 1-2	HTTP请求	356	✔	5963
9	03:54:21.182	User 1-4	HTTP请求	160	✔	5963
10	03:54:21.120	User 1-5	HTTP请求	237	✔	5963
11	03:54:21.194	User 1-8	HTTP请求	176	✔	5963
12	03:54:21.222	User 1-2	HTTP请求	161	✔	5963
13	03:54:21.168	User 1-3	HTTP请求	228	✔	5963
14	03:54:21.206	User 1-6	HTTP请求	258	✔	5963
15	03:54:21.343	User 1-4	HTTP请求	204	✔	5963
16	03:54:21.371	User 1-8	HTTP请求	189	✔	5963

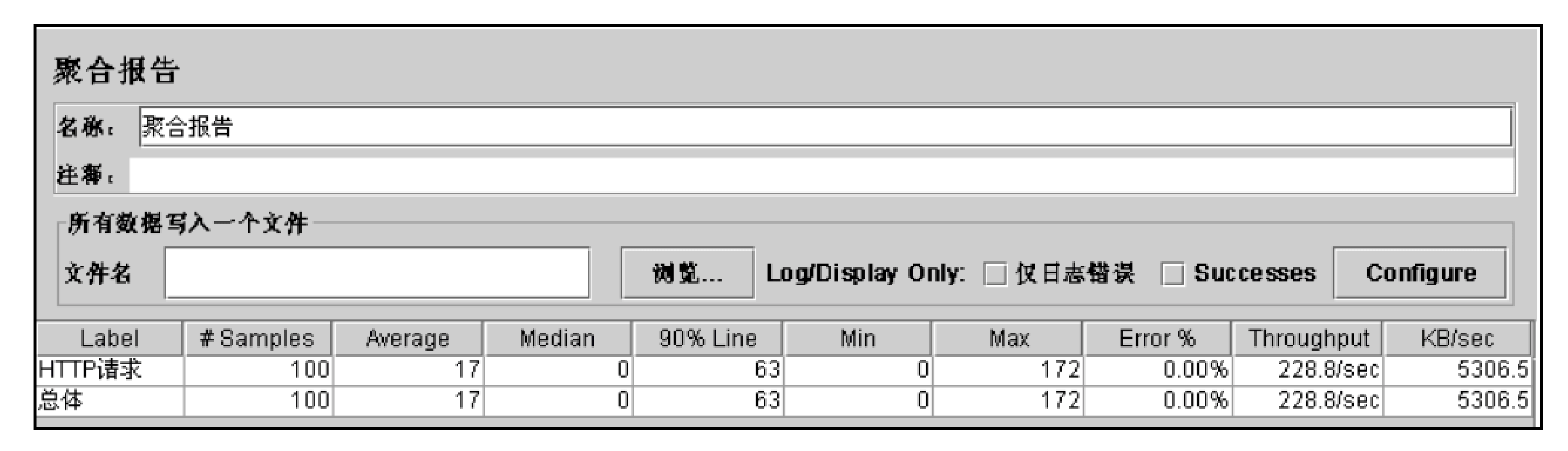
图 11-20　用表格查看结果

(3) 聚合报告结果如图 11-22 所示。

聚合报告标题解释：Label 用于说明请求类型，如 HTTP、FTP 等；#Samples 也就是图形报表中的样本数目，即总共发送到服务器的样本数目；Average 也就是图形报表中的平均值，它的值为总运行时间除以发送到服务器的请求数；Median 也就是图形报表中的中间值，是代表时间的数字，有一半的服务器响应时间低于该值而另一半高于该值；90%Line 是指 90%请求的响应时间比所得数值还要小；Min 是代表时间的数字，是服务器响应的最短时间；Max 是代表时间的数字，是服务器响应的最长时间；Error%是请求的错误百分比；Throughput 也就是图形报表中的吞吐量，这里是指服务器每单位时间处

理的请求数，注意查看是秒还是分钟；KB/s 是每秒钟请求的字节数。

图 11-21　查看结果树



Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
HTTP请求	100	17	0	63	0	172	0.00%	228.8/sec	5306.5
总体	100	17	0	63	0	172	0.00%	228.8/sec	5306.5

图 11-22　聚合报告结果

11.3.5　JMeter 工具小结

JMeter 是一款性能测试工具。与其说它是一个工具，不如说它是一个框架。因为 JMeter 的支持范围非常广，目前常见的需要进行性能测试的应用几乎都能应用(如 files、Servlets、Perl scripts、Java Objects、Data Bases and Queries、FTP Servers 等)。通过使用 JMeter 提供的功能，可以可视化地制订测试计划，包括规定使用什么样的负载、测试什么内容、传入的参数；同时，它还提供了多种图形化的测试结果显示方式，使用户能够更简便地开始测试工作和分析测试结果。JMeter 的一大好处就是它内部已经有实现好的线程机制，用户不用编写任何关于并发的东西，只需做简单配置即可。JMeter 还提供了一些类似插件的东西，用于线程运行时的控制。其次，JMeter 对测试结果能产生相应的统计报表，简单、直观，对于一般性能测试来说应该足够了。

实 验 习 题

1. 应用微软 WAS 工具进行 Web 应用的性能测试，详细给出测试过程和测试结果，并与 WebLoad 进行功能上的比较。

2. 应用 TPTP 工具对特定的 Web 应用进行性能测试，并与 JMeter 进行全面比较。

第Ⅵ部分　软件综合评测篇

前面的章节中介绍了有关软件测试的各种开源工具：①管理工具，如缺陷管理、测试管理等；②静态分析或测试工具，如程序理解、代码检查等；③单元测试工具，如单元“黑盒”测试、单元“白盒”测试等；④界面测试工具，如桌面软件、Web 应用等界面测试；⑤性能测试工具，如单元性能、系统性能(基于网络或基于 Web 应用的)。实际上，本书所涉及的工具仅仅是很少的一部分，当然我们认为它们是具有一定代表性和认知度的。可能读者已发现这些工具很零散、功能较弱，离全生命周期软件测试的要求或对软件全面评测的要求差距很大，以至于它们无法应用在对测试要求苛刻的大型的、复杂的和重要的软件测试中。特别是对于第三方的专业软件评测机构来说，希望有能对软件测试各种要求提供比较好支持的工具及集成解决方案，这也是为什么一些商业软件测试工具比较受欢迎的原因。诸如 IBM Rational/Telelogic、HP-Mercury、Compuware 以及 Parasoft 等公司完整的测试解决方案，包括测试管理、缺陷管理、功能测试、性能测试、覆盖测试、代码质量检查等测试内容及测试工具。但上述公司测试解决方案中的测试工具很多是单个或独立销售的，要全部配齐需要很大的资金及人力投入。现在国内外是否有公司开发了这么一个包含上述测试内容或要求的测试工具，以支持用户对软件进行综合测试？回答是肯定的。如商用的有 IBM Rational Logiscope、ISA 公司的 Panorama++；而开源的则有南京大学研发的 EASTT。

Panorama++是一个支持软件开发整个生命周期的软件工程与量化质量测评管理系统。

(1) 项目管理支撑：立项预估、规划、监控、结项。

(2) 需求阶段支撑：UML 建模分析、管理、与设计和源码间的自动追溯查错。

(3) 设计阶段支撑：大型系统的快速分层设计、源码框架生成、逆向工程。

(4) 编码阶段支撑：支持增量式、高一致性与低风险编码。

(5) 测试阶段支撑：包括以下内容。

① 基于复杂度分析的测试规划；

② 基于被测软件的全部源代码的可视化与自动链接的源代码审议和走查支撑；

③“黑盒”功能测试(采用线性脚本的动作自动录制与回放，有无源代码均可，二进制代码测试与所使用的计算机语言种类无关，基于源码的测试则与“白盒”结构测试无缝地相结合)；

④ 性能(动态用时分配)测试分析包括各程序分支的执行频度分析；

⑤“白盒”结构测试，支持美国和欧洲航空航天最高质量标准 RTCA/DO178B-LEVEL A 的 MC/DC(修改条件/判断覆盖)测试覆盖率分析；

⑥ Internet 应用程序的加载测试、分布式中间件与服务器端和客户端的应用程序的配对测试支撑；

⑦ 内存泄漏与违规使用检测、高效率测试用例设计支撑、测试用例有效性分析与测试用例最小等效集的自动生成。

(6) 度量与分析支撑：面向对象的个性化度量项的选择与度量标准的设定支持，动态与静态质量数据的自动收集、自动分析以及多形式的分析结果彩图显示。

(7) 维护阶段的支撑：动态运行错误自动定位、错误执行路径追溯、高一致性源码修改支撑、高一致性数据修改支撑。

(8) 配置支撑：版本维护与管理、智能化多级版本比较等。

南京大学承担和执行了共创软件联盟 863 项目中的《嵌入式应用软件测试技术》项目，研发了嵌入式应用软件分析测试工具 EASTT。EASTT 要求包括结构分析、质量分析、动态测试分析、嵌入式操作系统平台上应用软件的测试分析和文档生成等功能，起点高、要求高、水平也要高是研发该项目的最终目标，同时期望以先进成熟的且行之有效的技术作为开发技术，形成自己的特色和创新。该课题源码早期在联盟网站上对社会公开发布。

在本书中介绍该工具的宗旨是：①它是开源项目；②学习软件综合评测好的思想、方法和技术，并对其有一个全面了解；③对国产软件的厚爱及产品化的期望；④软件测试真的很重要、很有必要，本书给出的软件综合评测工具 EASTT 就是一个最有说服力的例子。

第12章　软件综合评测工具EASTT

软件评测是对软件产品的功能、性能文档资料、操作等进行相关的评价和测试，甚至包括对软件产品源代码的质量分析、度量和评价。由于软件重要性日益迫切，实时性强，软件评测已成为当前世界软件界的新兴产业。国内外大量的政府权威部门、企业重要部门以及独立的第三方均开始成立软件评测中心。这几年软件评测中心如雨后春笋般蓬勃发展。

软件评测中心对企业研发的软件产品进行评测，一方面可以发现软件产品的内部问题——以评为主(代码质量问题，如代码编写规范、代码复杂性、代码质量等；文档问题，如文档齐全、文档编写标准以及文档与代码的版本对应等)和外部问题——以测为主(功能、性能及结构等的测试)，提高软件的质量；另一方面可以证明所提供的软件质量符合标准规范要求，对软件的发布、使用以及市场营销起到良好的推动作用。因此，软件评测的目的就是为了保证软件产品的最终质量，促使企业在软件开发的过程中，对软件产品进行质量控制。

软件评测中心对用户委托的软件产品进行评测(如通过招投标来选择软件产品，请软件评测中心对投标的软件产品进行全面评测和综合评判)，可以帮助用户更好地了解该产品，为选择优质软件产品提供参考。

对于政府而言，依托软件评测中心，通过必要的监督、抽查和测试，及时掌握软件产品的质量状况，正确引导软件产业正常、有序和健康地发展。

一般软件评测中心依据GB/T 16260—2006《软件工程产品质量》和GB/T 17544—1998《信息技术软件包质量要求和测试》等标准，依据被评测软件对象的类型和特点，针对6大质量特性、21个质量子特性，选择不同的度量元，形成有针对性的评价体系，并以此为依据对被测对象进行质量评测。

事实上，软件评测中最核心的工作还是软件测试。软件测试就是在受控制的条件下对系统或应用程序进行操作并评价操作结果的过程，所谓控制条件应包括正常条件与非正常条件。软件测试过程中应该故意地去促使错误发生，也就是事情在不该出现的时候出现或者在应该出现的时候没有出现。从本质上说，软件测试是“探测”，在“探测”中发现软件的毛病。软件测试包含“白盒”测试与“黑盒”测试，“白盒”测试是针对程序代码进行正确性检验的测试工作；“黑盒”测试则是独立于程序代码，从用户的角度，通过一定的测试步骤与测试用例，验证软件功能、性能等指标能否满足实际应用需求的测试工作。

一般来说，软件测试应由独立的软件评测中心负责，严格按照软件测试流程，制订测试计划、测试方案、测试规范，实施测试，对测试记录进行分析，并根据回归测试情况撰写测试报告。测试是为了证明程序有错，而不能保证程序没有错误。

软件评测中心担负着软件评测的重任，为了完成评测工作，他们必须配备各种软件评测工具(当然也要全面地掌握软件评测方法和技术)。那么如何选型呢？南京大学计算机科学与技术系软件开发环境与方法组研制并开发的 EASTT 工具集成了目前较为流行的软件分析测试方法，应用了各种先进的软件评测技术，学习和使用 EASTT 会对我们理解并掌握软件评测技术起到抛砖引玉的作用。

12.1　EASTT 工具介绍

南京大学计算机科学与技术系承担的《嵌入式应用软件测试技术》作为《嵌入式应用软件平台》的一个组成部分被列为 863 联盟项目，成为我国首批在联盟网站开放的自由软件之一，这便是 EASTT 的雏形。在 863 联盟项目说明会之后，项目组认真讨论这一新形式的任务。通过讨论，大家达成一个共识，一定要高要求、高起点、高水平地完成 EASTT 软件开发任务。第一，以实现 Logiscope 和 Panorama C++的功能为最低目标。Logiscope 是当时法国 VERILOG 公司开发的面向嵌入式应用的关于 Visual C++的分析测试工具，目前在包括中国在内的国际市场上占有很大的市场份额。Panorama C++是美国 ISA(International Software Automation)公司的软件分析测试产品。以这两个同类产品的功能作为最低实现目标，就可以使 EASTT 在系统功能上达到国际先进水平。第二，以先进成熟的且行之有效的技术作为开发技术。这样就能确保开发工作在一个既先进又有效的基础上进行。第三，要有特色和创新。在达成此共识的基础上，嵌入式应用软件分析测试工具 EASTT 便应运而生了。

因此可以说，EASTT 是南京大学计算机科学与技术系软件开发环境与方法组研制并开发的一个嵌入式应用软件的集成化分析测试工具包。该工具包主要研究了对嵌入式面向对象应用软件进行分析测试的先进技术和进行质量评价的科学方法。EASTT 是针对 C/C++语言的测试工具集，可用于嵌入式应用软件的静态结构分析、动态测试跟踪。它还提供了一个层次化模型来对应用软件的质量进行综合评价。

1. EASTT 系统的主要研究内容

主要研究内容包括：①对嵌入式面向对象应用软件进行分析和测试的先进建模技术；②对测试技术带来的开销进行控制的有效技术；③用于嵌入式应用软件的各种测试方法；④面向对象软件的度量体系和质量评价方法；⑤开发嵌入式应用软件的功能强、水平高的结构分析工具、测试分析工具、软件质量分析工具和文档资料自动生成工具，并把它们集成为一个有机的整体。

2. EASTT 系统的主要功能模块

主要功能模块包括：静态分析器、动态插桩器、测试分析器、质量模型编辑及分析器、结果浏览器和文档生成器。其中，静态分析器是用 Lex 和 Yacc 工具生成的语法分析

工具和语义分析工具，用于对源程序进行语法分析和语义分析，在此基础上由中间结构管理器构建中间结构，进行静态分析，并将中间结构和静态分析结果存放在二进制文件中；而动态插桩器则是在中间结构的基础上进行插桩，产生插桩文件。插桩文件和 RTL 库在开发环境下编译连接成可测试程序。

3. EASTT 系统的主要应用

1) 用于软件的开发阶段

定义软件质量度量模型：利用 EASTT 提供的图形化软件质量度量模型编辑器，企业可定义自己内部软件质量管理用的符合 ISO 9126 标准的(Metrics_Criteria_Factor)三级软件质量度量模型。确认和改进设计及代码：利用 EASTT 提供的结构分析和质量分析功能，程序员可以检查和确认自己的设计及编码的质量，尽早检测出关键部分，对不符合质量要求的关键程序段进行修改。

2) 用于软件的测试阶段

定义测试准则：利用 EASTT 提供的用于测试用例的执行路径分析和覆盖分析等，项目主管可定义要达到的覆盖率，以判断何时结束测试阶段。度量测试的有效性：利用 EASTT 提供的测试覆盖信息和图表，测试人员可随时得知测试用例的有效性。优化测试过程：利用 EASTT 提供的测试用例集最小化分析，可对运行过的测试用例进行优化。

3) 用于软件的审核

利用 EASTT 提供的结构分析和质量分析功能，项目主管可随时检查和审核项目的进展情况和质量。

4) 用于软件的维护和逆向工程/再工程

EASTT 提供的结构分析和文档生成等功能，可以帮助维护人员和逆向工程/再工程人员理解被测软件，并利用结果进行维护和再工程。

5) 用于帮助开发人员编写软件的文档资料

6) 用于支持宿主机及嵌入式操作系统平台上的应用软件测试

EASTT 支持 Host/Target 实时交叉测试方式。目前支持的嵌入式操作系统是 pSOS。

4. EASTT 的主要特色

1) EASTT 是层次化的、全面可度量的

EASTT 包括对高层的类继承性及多态性的分析，对类/函数的耦合关系及引用关系的分析，以及对最底层的类/函数的内部结构分析。

EASTT 对被测软件既进行静态的结构分析、关系分析、流程分析、数据分析，又可以对动态的测试用例执行路径分析、效率分析、覆盖分析、数据分析、测试用例集最小化分析。

EASTT 对被测软件的各种重要基本成分进行度量，给出度量准则和度量因素，并提供用户编辑度量标准的手段。

2) EASTT 支持测试开销可计算、测试粒度可控制、测试结果可组装等测试要求

进行测试分析工作必然要占用许多运行时间。为了让用户了解程序的真实效率，提供对插桩测试开销进行计算和控制的有效手段及技术，是十分必要和有用的。EASTT 提供了各种灵活有效的控制手段，使得测试开销可计算、测试粒度可控制、测试结果可以汇总。

EASTT 设计开发了能适应不同测试环境的 Host/Host 和 Host/Target 两种测试方式。Host/Host 方式是针对测试分析工具和被测软件在同一台机器上运行的情形。Host/Target 方式则是针对测试分析工具和被测软件在不同机器上运行的情形。

EASTT 构建了三级软件质量度量模型，提供了该模型的编辑器。设计开发了关于 OOPL 的各种重要成分的基本度量元，并按照 ISO 9126 标准构建了三级软件度量模型——基本度量元级(Metrics)、度量准则级(Criterion)和度量因素级(Factor)。还为用户提供了使用方便的图形化的度量准则编辑手段。

EASTT 提出并实现了易于复用的独立于 OOPL 的层次语义模型 HSM。HSM 主要由类语义模型层、函数语义模型层和控制流模型层组成。根据 HSM 可以得出需要的各种分析信息。

12.2　EASTT 测试环境建立

EASTT 由 C++实现，EASTT 的开发者仅在 Microsoft VC++ 6.0 下进行了简单测试。

由于库函数不同，EASTT 的源代码不能使用更高版本的 Microsoft Visual Studio 进行编译，否则会出现语法错误。

EASTT 在遵循源码许可证 GNU GPL 的条件下可以用于任何评估及教学目的。由于环境和设备不同，使用人员可能需要对 EASTT 做一些修改。

EASTT 可在共创软件联盟网站下载(可能网站已关，本教材随盘附有)。EASTT 以 zip 压缩格式发布，其相关文档是.ps 格式。EASTT 目前仅支持 Windows 9x、Windows NT 4.0、Windows 2000 平台环境。解压后可以直接使用，但有些功能在使用过程中可能会出现问题，需重新编译。

安装 EASTT：将 Eastt.zip 解压缩到 EASTT 目录下，对安装路径没有特殊要求。

卸载 EASTT：直接删除整个目录即可。安装后的 EASTT 的整体目录结构如表 12-1 和表 12-2 所示。

表 12-1　EASTT 目录结构

主要文件夹	功　　能
gamma 通信库	提供动态测试测试用例所需的文件
Source	ESATT 软件的源代码
sample	提供几个 C/C++的可测试样例
lib	提供预处理所要使用的不同语言的库

表 12-2　Source 文件夹下的主要文件夹和源码文件

主要文件夹	功　　能
Doc	包含若干.ps 格式的文档，提供有关系统的使用说明。这些文档随 EASTT 一起发布，并随版本同步更新
Ooa	有关对被测程序进行语法分析的源文件
Ism	有关对被测程序进行中间结果表示的源文件
Oot	有关对被测程序进行测试分析的源文件
Oov	有关系统总控和分析结果显示的用户界面源文件
Sqm	包含几个用户可自定义的质量模型定义文件

一般情况下，开发者无须变动以上目录结构。

用 Microsoft Visual Studio 6.0 打开 Source 目录下的工作区，进行调试和重新编译，主要会遇到以下问题。

问题一：找不到头文件。

原因：没有设置 include 路径。

解决办法：设置路径，依次单击“工具”|“选项”下的“目录”标签，双击添加路径，路径为安装 EASTT 软件的 Source 文件夹，如图 12-1 所示。

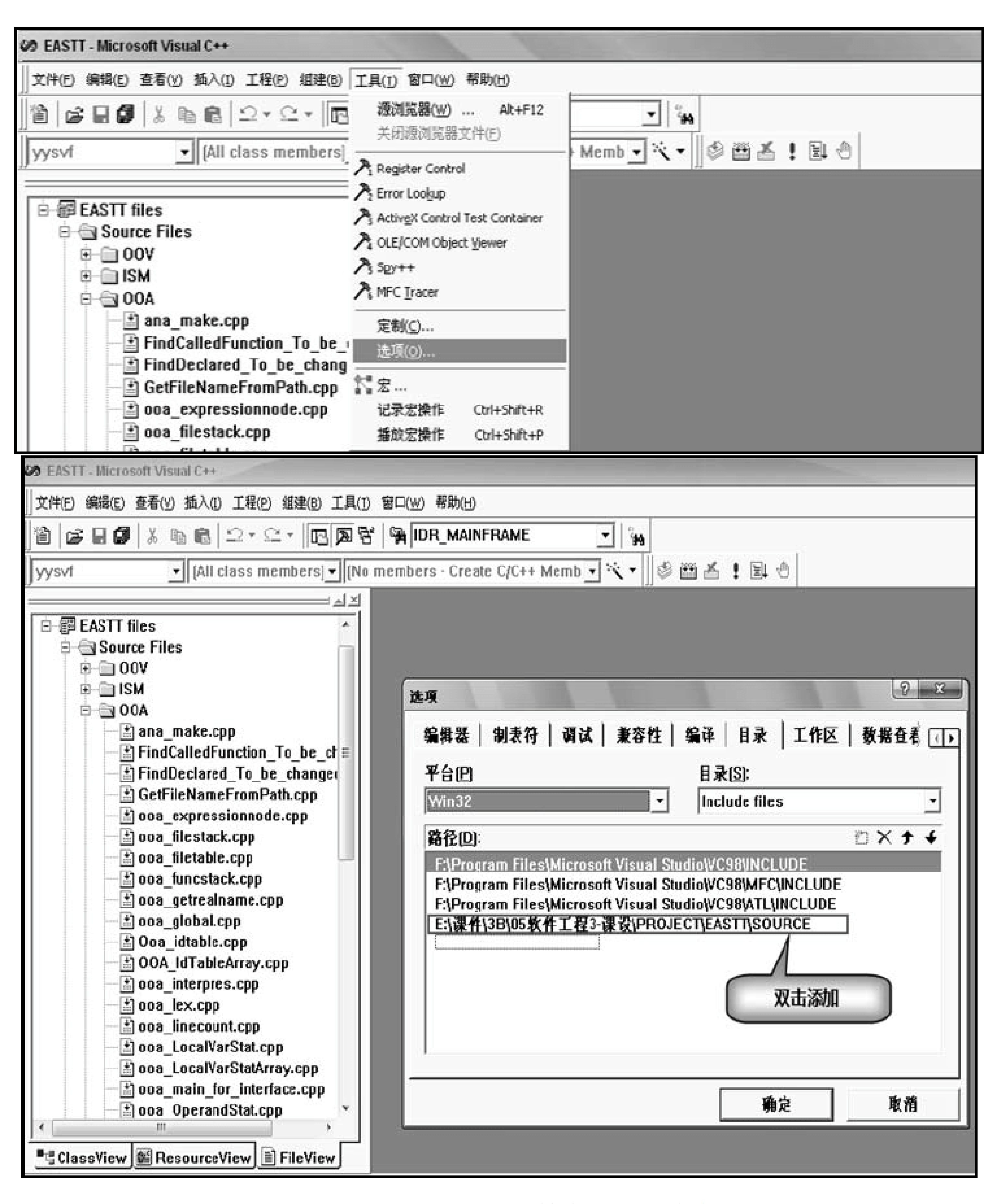


图 12-1　指定头文件的目录路径

问题二：仍然有一些头文件找不到。比如“fatal error C1083: Cannot open include file: 'ooa_interpres.h': No such file or directoryInsertLevelDlg.cpp”。

原因：解决问题一的时候，将头文件的路径设为了 Source 文件夹，而由于许多头文件在 Source 文件夹的子文件夹中，因而系统在 Source 文件夹中查找这些文件时，自然就找不到了，就会报错。

解决办法：找到报错处，将路径添加完整。比如将源码中的#include "ooa_interpres.h"改为#include "OOA/ooa_interpres.h"。

问题三：找不到文件 EasttSysDirectoryDlg.cpp。

原因：组建系统时，需要组建这个文件，而实际上 Source 文件夹下并不存在这个文件，所以组建时出错。

解决办法：要解决这个问题有两种办法。这个文件并不影响整个系统的运行，不起实际作用，所以一种方法是可以在 Source 文件夹下新建一个空文件，命名为 EasttSysDirectory Dlg.cpp，再进行组建就可以通过了。还有一种解决办法就是删掉 EASTT.plg 文件。

问题四：在未安装 VC 6 环境的情况下直接运行可执行文件 EASTT.exe 时会提示缺少 MFC42D.dll 和 MSVCRTD.dll。解决方法是将这两个缺少的 DLL 文件复制到%system%\windows\system32 中。

解决以上 4 个问题后，应该就可以编译、组建通过了。

运行结果如图 12-2 所示。



图 12-2　EASTT 主界面

在 debug 文件夹中运行 EASTT.exe 出现无 Precomp.dll 的错误，其解决方法是将 Source 目录下的 Precomp.dll 复制到 c:\windows\system32 文件夹下。

12.3　EASTT 测试功能及使用流程

在介绍 EASTT 的主要功能之前，在这里首先要强调一下，尽管 EASTT 是针对嵌入式应用开发的，但对于非嵌入式应用它也是非常适用的。

EASTT 的主要功能如下。

(1) 对被测软件的结构进行静态分析，然后用统计图表、直方图、类继承图、类耦合图、函数调用关系图、层次流程图等形式将分析结果直观地显示出来。

(2) 对被测软件的动态行为进行分析，并针对不同的测试用例分析多种级别下的程序

覆盖情况。

(3) 通过一组适用于面向对象编程的度量元，采用三级质量度量模型，对被测软件的质量从多个角度进行评估。

(4) 针对嵌入式操作系统平台上的应用软件，提供测试分析的工具。

(5) 自动生成被测软件系统的测试报告，并以多种形式将测试结果反馈给用户。

图 12-3 全面描述了应用 EASTT 进行软件评测的完整流程。

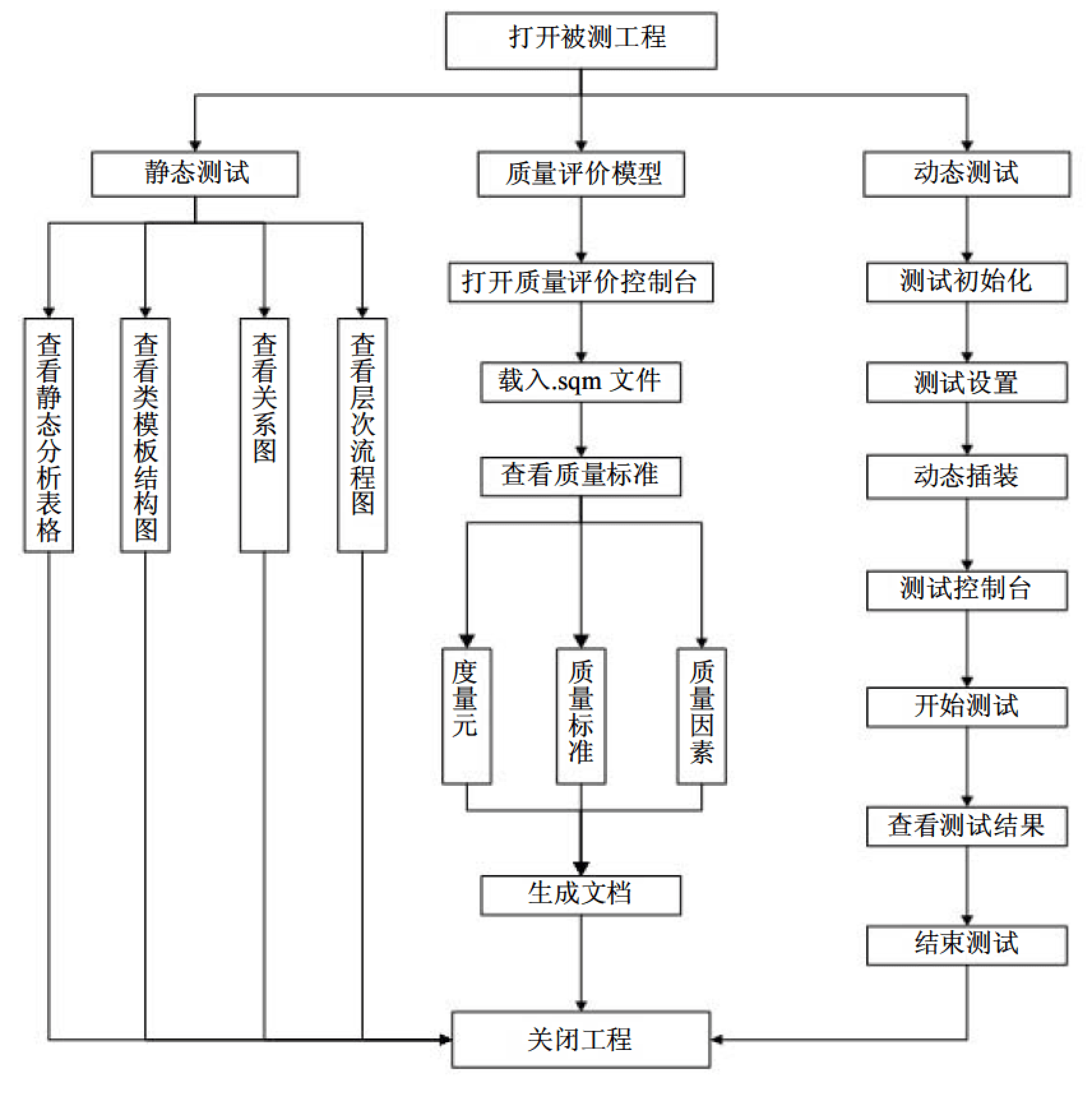


图 12-3　EASTT 的使用流程

下面就上述 EASTT 的功能和使用流程做详细介绍。

12.3.1　EASTT 的主要功能

1. 被测软件的结构分析

EASTT 可提供：①可浏览源代码的类表；②反映继承关系的类继承图；③反映函数调用关系的函数调用图；④反映类-函数间耦合关系的类-函数调用图；⑤反映函数内部流程的可伸缩的层次流程图。

2. 被测软件的质量分析

EASTT 可提供：①符合 ISO 9126 标准的三级软件质量度量图表；②OOPL 各种重要

成分的分析统计数据报表；③可视化的质量评估用的 Kiviat 图、直方图和饼图；④直观的图形化软件质量度量模型编辑器。

3. 被测软件的动态测试分析

EASTT 可提供：①测试用例的执行路径分析；②带有覆盖信息的测试层次流程图；③带有覆盖信息的函数调用图；④带有测试分析信息的软件质量度量图表(Kiviat 图、直方图、饼图)；⑤测试用例的效率分析；⑥测试用例集最小化分析。

4. 嵌入式操作系统平台上应用软件的测试分析

EASTT 可提供：①计算测试开销；②控制测试粒度；③汇总测试结果；④支持 Host/Target 的实时交叉测试方式。

5. 被测软件的文档生成

EASTT 可提供自动组织和生成以上多种分析测试结果的文档资料。

12.3.2　EASTT 的使用流程

1. 启动软件，打开被测项目

在 EASTT 软件目录下双击 EASTT.exe 以启动软件，在菜单栏选择“文件”|“打开”命令或者直接单击工具栏上的“打开”图标，在出现的对话框里定位要进行分析的 C++程序并将其打开。EASTT 只支持用 Visual C++ 6.0 开发的 C++程序，故打开的是 VC 6 的项目文件(.dsp)。

注意：*工程的.dsp 文件后缀名一定是小写字母，否则打开工程文件时会报错。另外，要设置系统路径。系统路径设置为 EASTT 软件安装目录。*

打开项目文件后，EASTT 的界面并不会有太大变化，但是菜单栏和工具栏上的可选项增多了，这表明已经可以对该项目进行各种分析了，如图 12-4 所示。

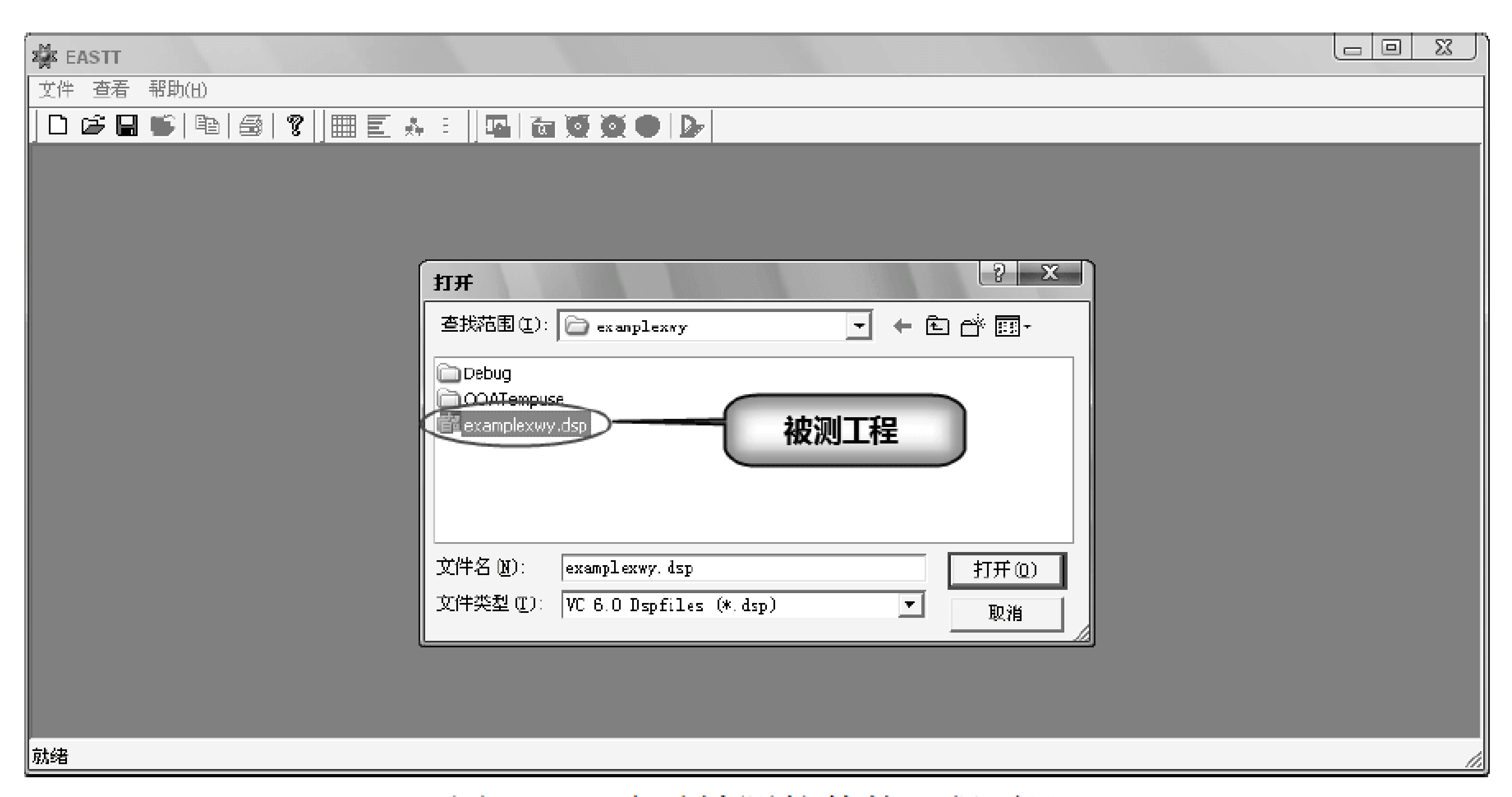


图 12-4　启动被测软件的工程项目



图 12-4　(续)

2. 静态结构分析

EASTT 静态结构分析提供对被测试工程多种方式的静态结果浏览，主要包括静态分析表格、结构浏览图、关系图、层次流程图。它依靠静态分析器来完成这一工作，主要是分析源代码中函数与函数之间的关系，类与类之间的关系，以及源代码的文件类型、行数等。EASTT 静态测试采用的是新一代软件静态分析技术—— 抽象语法树(Abstract Syntax Tree，AST)，并附加了丰富的对语义状态的描述。

1) EASTT 静态分析表格

当用户选择“静态分析”|“静态分析表格”菜单项时，工作区中将产生一个分隔窗口。窗口左栏为一树型结构，静态分析的所有基础指标都被分类安排在树型结构中；窗口右栏为一列表结构，当用户在左栏选中某个指标后，该列表结构就会显示出相应指标的详细内容。

(1) 文件相关信息列表：包括源文件汇总表(Source File Summary Table)(列出文件名、行数、最后修改时间属性)和文件概要信息表(File Compact Table)(列出代码总行数目、空行数目、注释行数、有效行数等属性)，如图 12-5 所示。

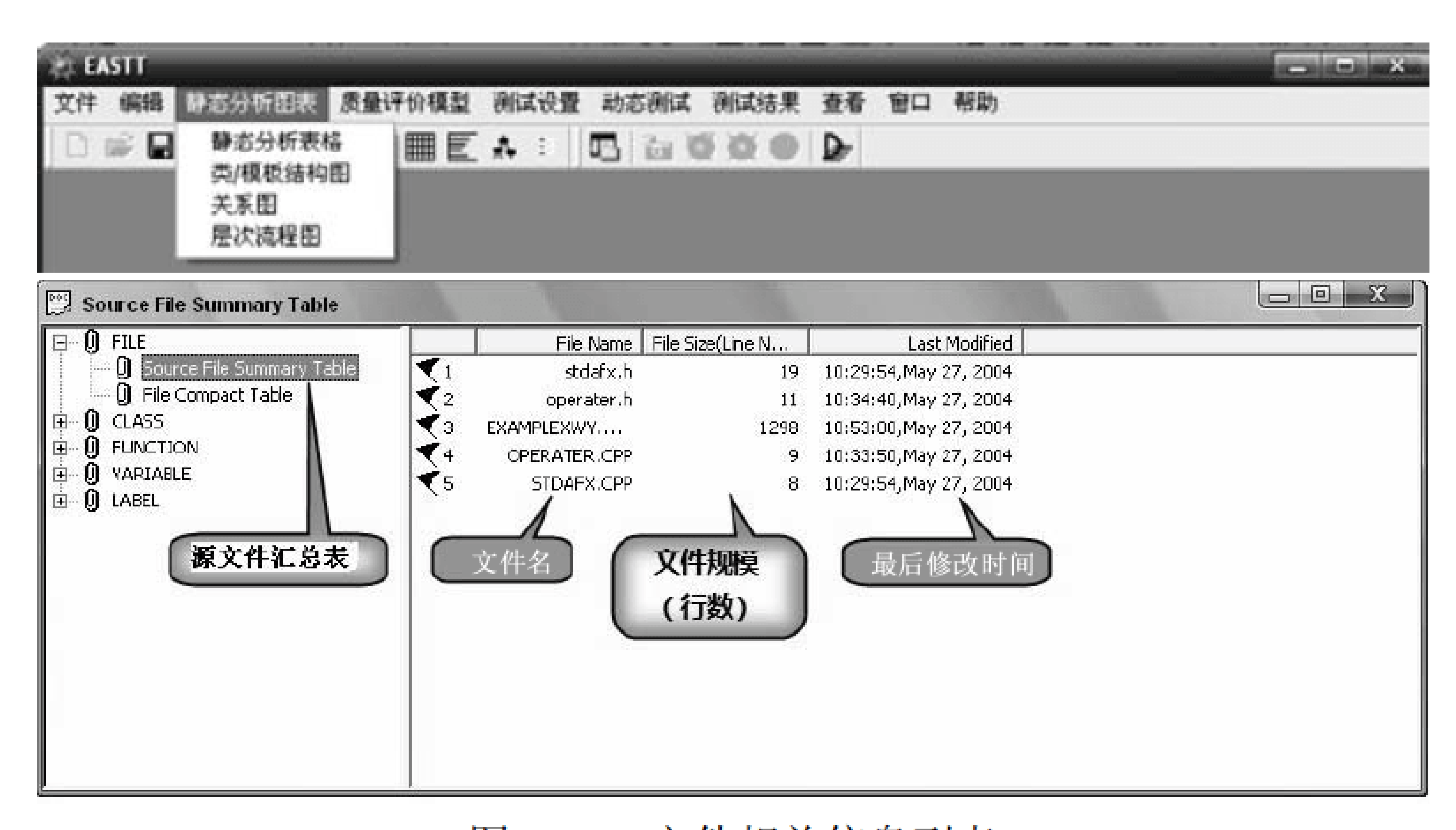


图 12-5　文件相关信息列表

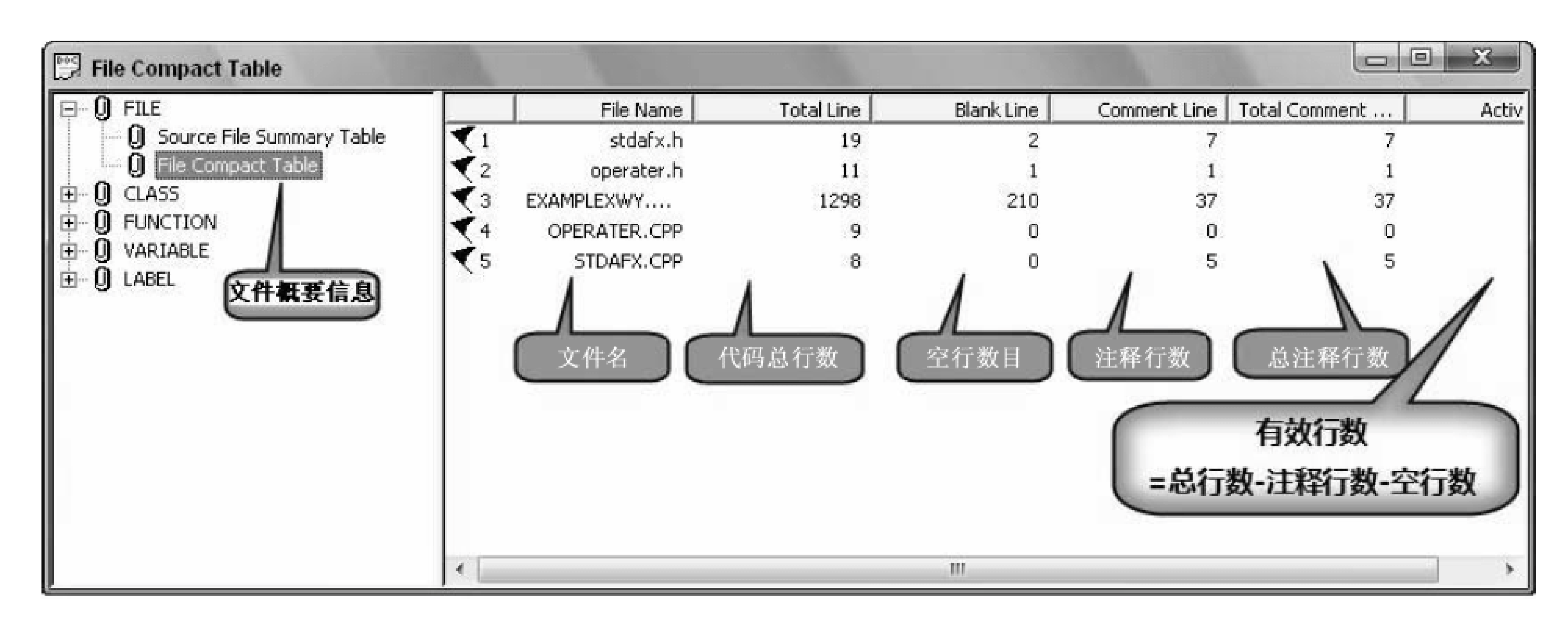


图 12-5　(续)

(2) 类相关属性表：包括类定义表(Class Definition Table)(列出类名、定义这个类的文件名、定义语句所在的行数)，类交叉引用：派生源(Class Cross-Reference: Derived From)(列出类名、父类、派生类型)，构造函数、析构函数表(Constructor Destructor Table)(列出类名、函数名、所在文件名、起始行数)，方法成员表(Method Member Table)(列出函数名、所在文件名、起始行)，友元类表(Friend Class Table)(列出类名、友元类名)等，如图 12-6 所示。

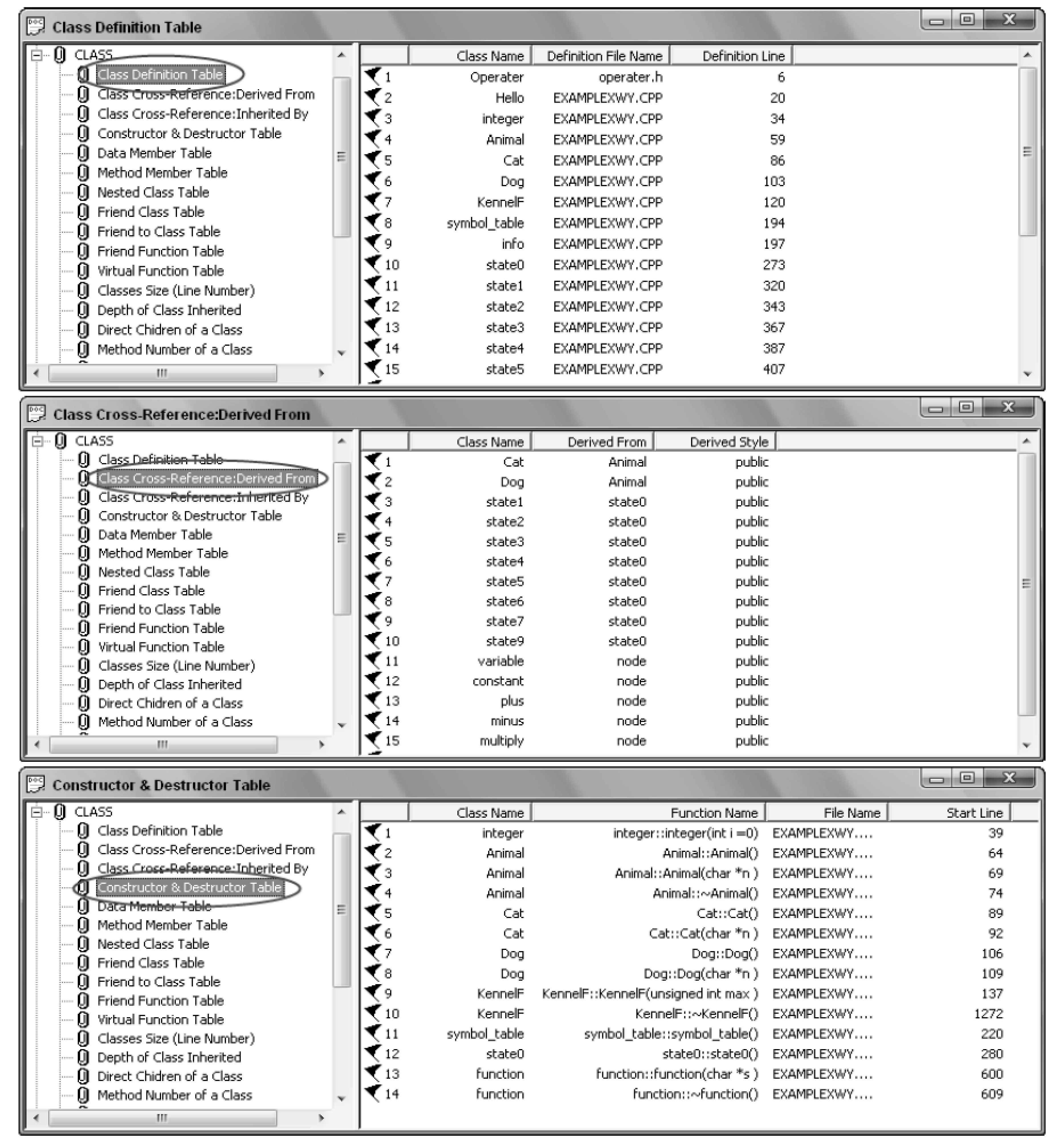


图 12-6　类相关属性信息列表

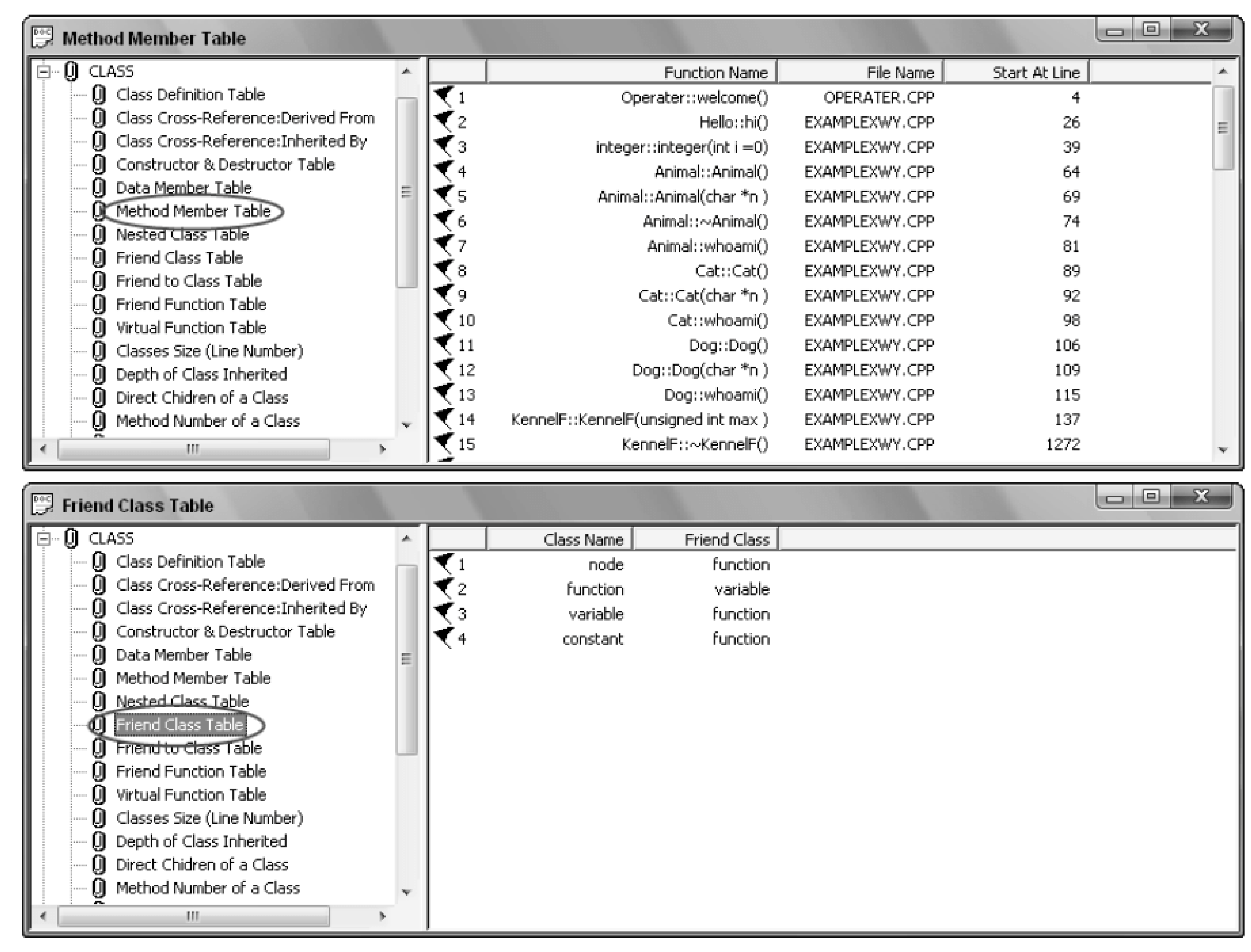


图 12-6　(续)

(3) 函数相关属性列表：包括函数定义表(Function Definition Table)(列出函数名、所在文件名、起始行)，未被调用函数表(Function Not Called Table)(列出未被调用的函数名、所在文件名、起始行)，重载函数表(Overloaded Function Table)(列出重载的函数名、所在文件名、起始行)，函数概要信息表(Function Compact Table)(列出函数名、所在文件名、总行数、空行、注释行、总注释行、有效行)等，如图 12-7 所示。

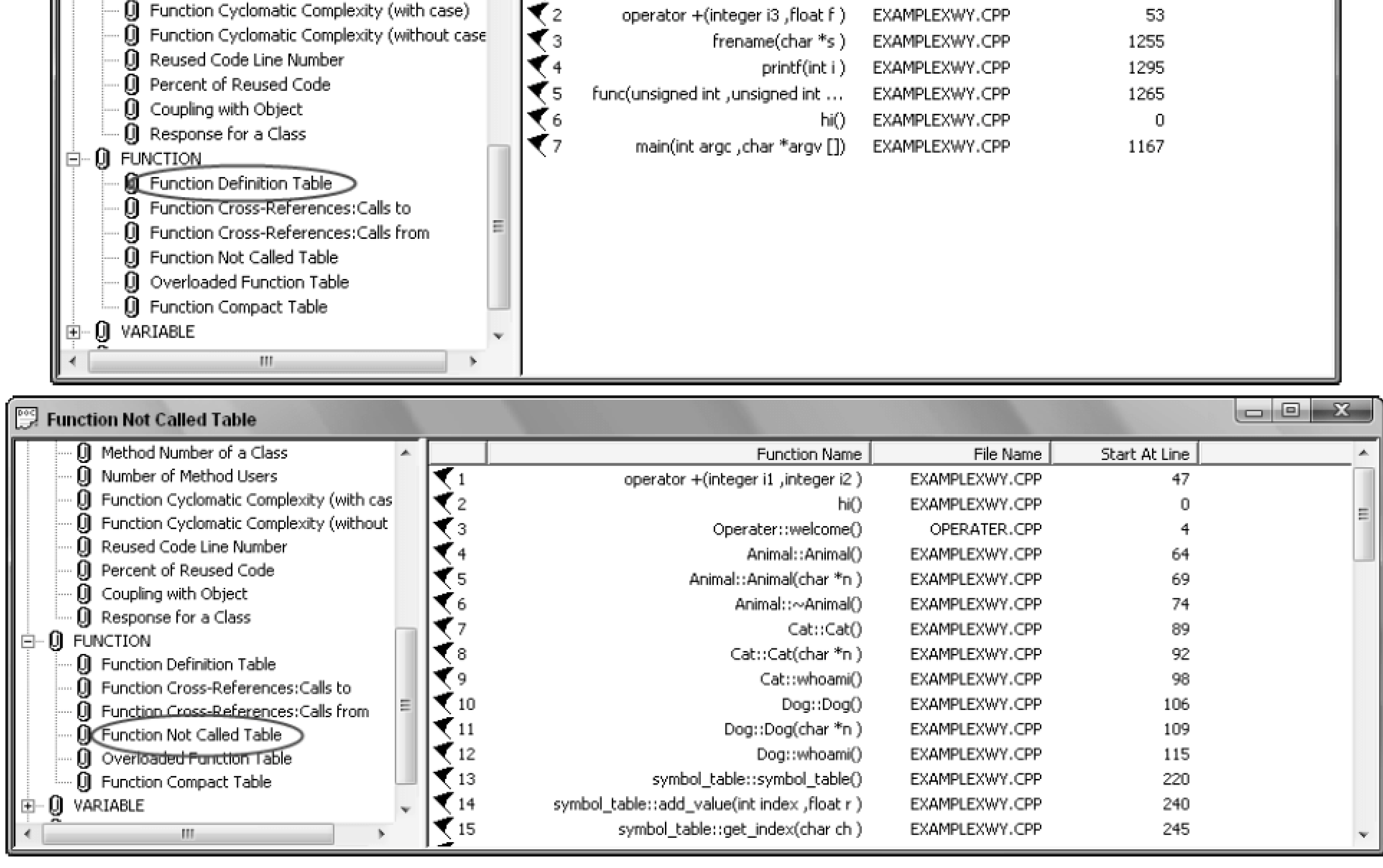
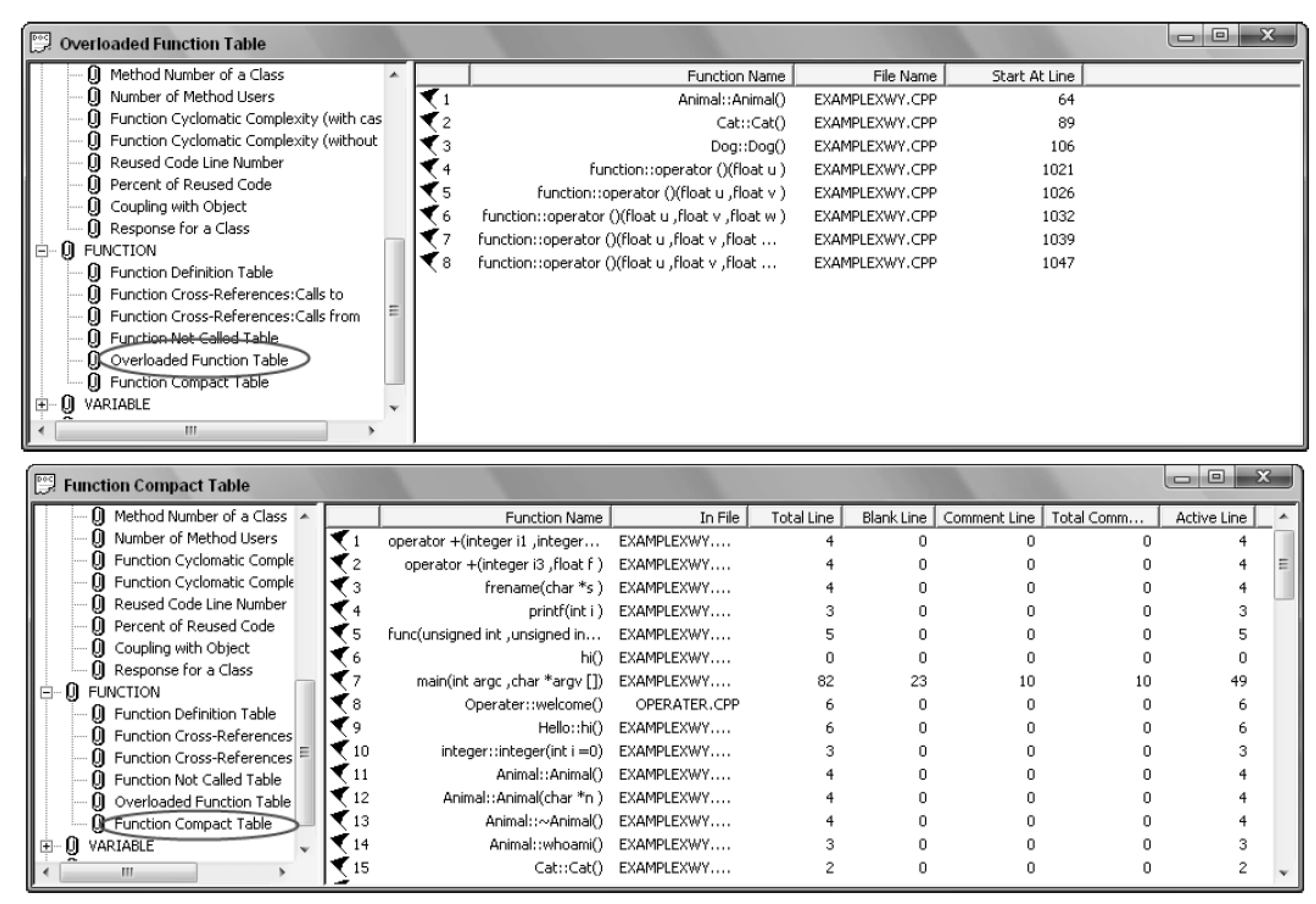


图 12-7　函数相关属性信息列表

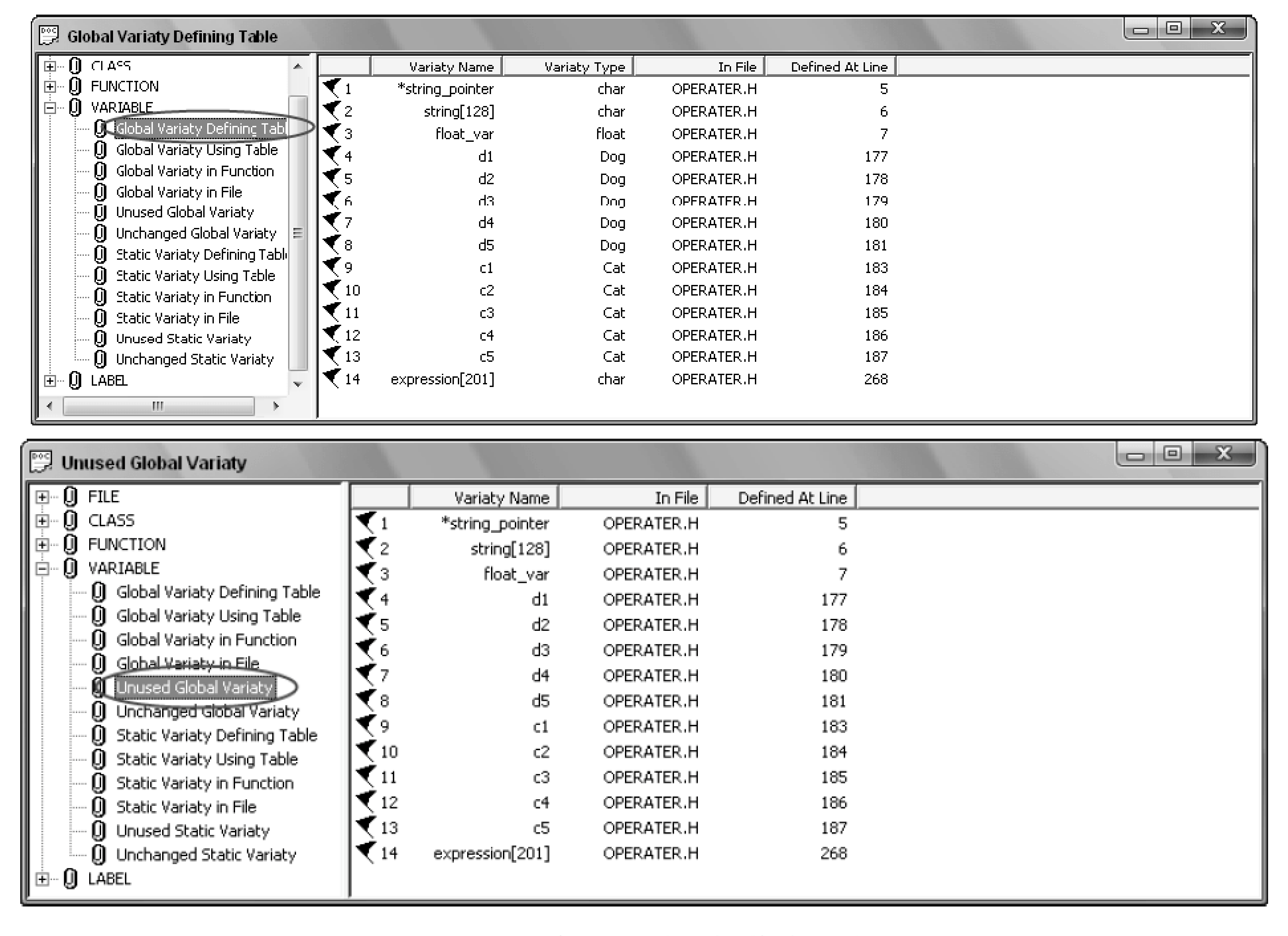


	Function Name	File Name	Start At Line
1	Animal::Animal()	EXAMPLEXWY.CPP	64
2	Cat::Cat()	EXAMPLEXWY.CPP	89
3	Dog::Dog()	EXAMPLEXWY.CPP	106
4	function::operator ()(float u)	EXAMPLEXWY.CPP	1021
5	function::operator ()(float u ,float v)	EXAMPLEXWY.CPP	1026
6	function::operator ()(float u ,float v ,float w)	EXAMPLEXWY.CPP	1032
7	function::operator ()(float u ,float v ,float ...	EXAMPLEXWY.CPP	1039
8	function::operator ()(float u ,float v ,float ...	EXAMPLEXWY.CPP	1047

	Function Name	In File	Total Line	Blank Line	Comment Line	Total Comm...	Active Line
1	operator +(integer i1 ,integer...	EXAMPLEXWY....	4	0	0	0	4
2	operator +(integer i3 ,float f)	EXAMPLEXWY....	4	0	0	0	4
3	frename(char *s)	EXAMPLEXWY....	4	0	0	0	4
4	printf(int i)	EXAMPLEXWY....	3	0	0	0	3
5	func(unsigned int ,unsigned in...	EXAMPLEXWY....	5	0	0	0	5
6	hi()	EXAMPLEXWY....	0	0	0	0	0
7	main(int argc ,char *argv [])	EXAMPLEXWY....	82	23	10	10	49
8	Operater::welcome()	OPERATER.CPP	6	0	0	0	6
9	Hello::hi()	EXAMPLEXWY....	6	0	0	0	6
10	integer::integer(int i =0)	EXAMPLEXWY....	3	0	0	0	3
11	Animal::Animal()	EXAMPLEXWY....	4	0	0	0	4
12	Animal::Animal(char *n)	EXAMPLEXWY....	4	0	0	0	4
13	Animal::~Animal()	EXAMPLEXWY....	4	0	0	0	4
14	Animal::whoami()	EXAMPLEXWY....	3	0	0	0	3
15	Cat::Cat()	EXAMPLEXWY....	2	0	0	0	2

图 12-7　(续)

(4) 变量相关属性列表：包括全局变量定义表(Global Variety Definition Table)(列出变量名、变量类型、所在文件、定义所在行)，未使用的全局变量(Unused Global Variety)(列出变量名、所在文件、定义所在行)等，如图 12-8 所示。



	Variaty Name	Variaty Type	In File	Defined At Line
1	*string_pointer	char	OPERATER.H	5
2	string[128]	char	OPERATER.H	6
3	float_var	float	OPERATER.H	7
4	d1	Dog	OPERATER.H	177
5	d2	Dog	OPERATER.H	178
6	d3	Dog	OPERATER.H	179
7	d4	Dog	OPERATER.H	180
8	d5	Dog	OPERATER.H	181
9	c1	Cat	OPERATER.H	183
10	c2	Cat	OPERATER.H	184
11	c3	Cat	OPERATER.H	185
12	c4	Cat	OPERATER.H	186
13	c5	Cat	OPERATER.H	187
14	expression[201]	char	OPERATER.H	268

	Variaty Name	In File	Defined At Line
1	*string_pointer	OPERATER.H	5
2	string[128]	OPERATER.H	6
3	float_var	OPERATER.H	7
4	d1	OPERATER.H	177
5	d2	OPERATER.H	178
6	d3	OPERATER.H	179
7	d4	OPERATER.H	180
8	d5	OPERATER.H	181
9	c1	OPERATER.H	183
10	c2	OPERATER.H	184
11	c3	OPERATER.H	185
12	c4	OPERATER.H	186
13	c5	OPERATER.H	187
14	expression[201]	OPERATER.H	268

图 12-8　变量相关属性信息列表

(5) 标号相关属性列表：包括标号定义表(Label Definition Table)(列出标号名、定义所在的文件名、定义所在行)，Goto 语句位置表(Goto Location Table)(列出标号名、定义所在的文件名、Goto 所在行)，未使用的标号表(Unused label Table)(列出标号名、定义所在的文件名、标号所在行)等。

2) EASTT 结构浏览图

当用户选择“静态分析”|“结构浏览图”菜单项时，工作区将产生一个分隔窗口。窗口左栏为一树型结构。该树型结构有两层，第一层列出所分析项目中的所有类和 Global 指示，第二层为类和 Global 的所有数据成员和函数成员，对每一个成员的属性(public、protected、private、friend 等)用相应的图标标识；窗口右栏为一文本窗口，当用户在左栏选中某个类或它们的函数成员、友元成员时，文本窗口中会用伪码显示出对应类的定义及函数成员的函数体。子类从父类继承的所有成员也将函数显示在该子类对应的第二层中。

结构浏览图主要是用于分析类中的方法和属性以及全局函数，可以把类中关于方法和全局方法的详细信息从源代码中摘列出来，如图 12-9 所示。

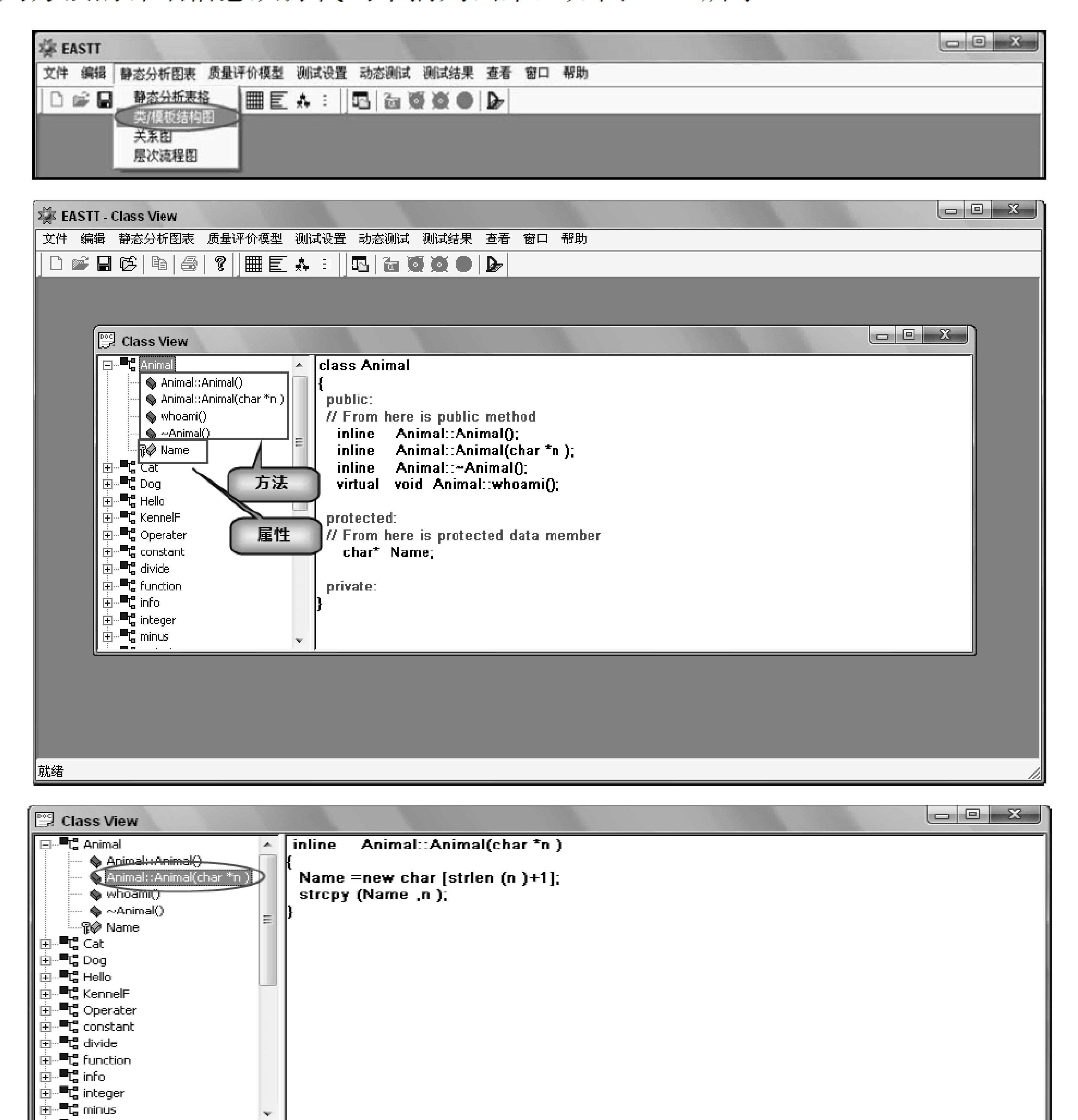


图 12-9　类中成员的声明及方法的具体实现

3) EASTT 关系图

EASTT 关系图着眼于揭示被分析项目中各基本单元的复杂关系。EASTT 主要选择了类继承关系、函数调用关系、类/函数耦合关系进行分析，并给出了相应的关系图。

当用户选择“静态分析图表”|“关系图”菜单项时，EASTT 将打开如图 12-10 所示的对话框，可由用户选择浏览哪一种关系图，并选择某个类或函数，选中后单击“打开”按钮，工作区将产生一个滚动窗口，图形化地显示选中单元所相应的关系图。

关系图主要是用来分析类继承、函数调用和类/函数耦合，在如图 12-10 所示的对话框中用户还可以选择关系图的显示模型是树型还是对称型。



图 12-10　“关系图”对话框

下面介绍一下 EASTT 关系图的基本概念和功能。

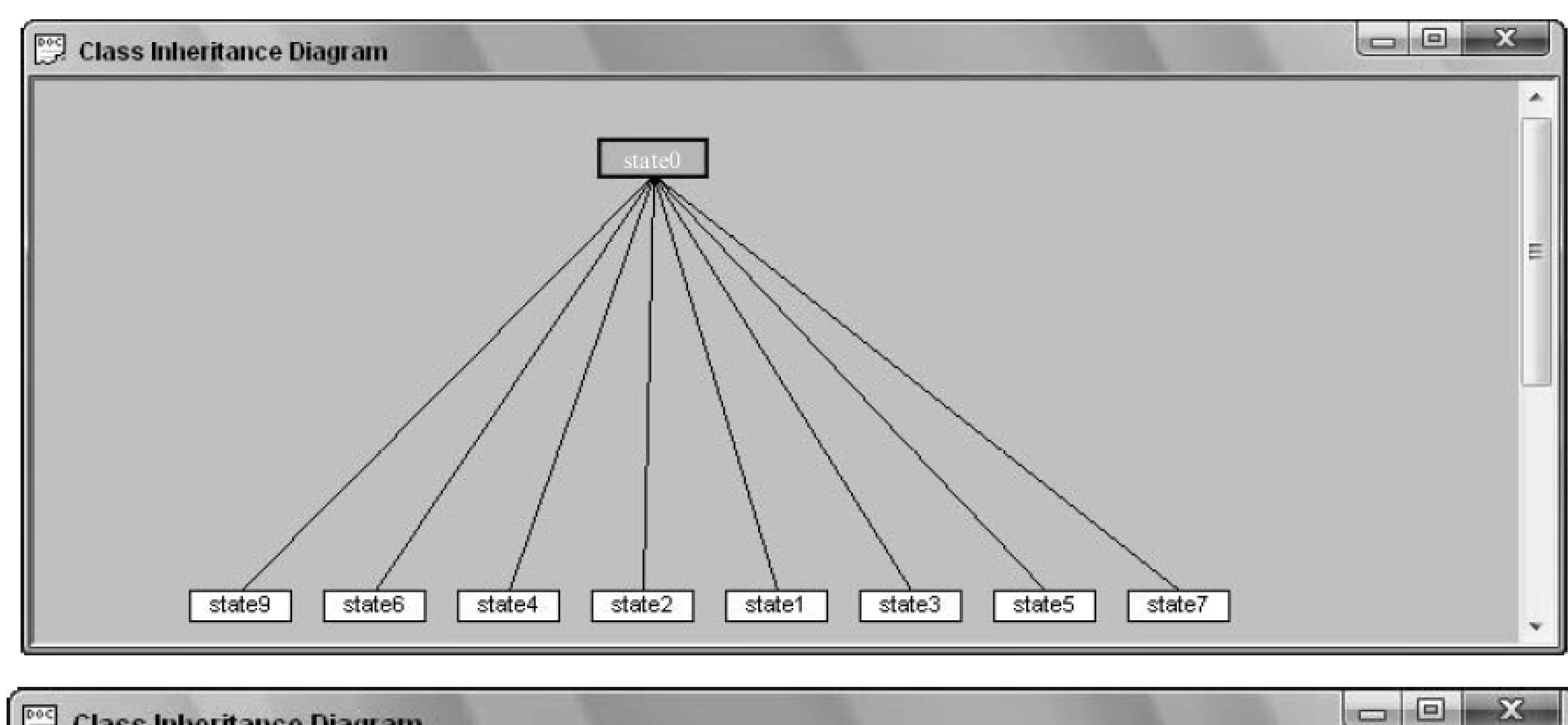
基本概念：一个节点对应一个基本单元(类或函数)，绿色显示的为当前节点，即用户当前选择的节点；若任意两个节点之间有用户要求的关系，则用一定的线条连接两节点。一般情况下，使用黑线连接节点；若这两个节点不在相邻层中，则用紫红色线连接两节点；若这两个节点是函数调用中的递归调用关系，则用一定角度的红色折线连接。

功能：EASTT 关系图支持对当前节点的自由切换功能。如果用户希望浏览图中的某一个非当前节点的关系图，可单击该节点，该节点将同时变为当前节点。EASTT 关系图提供类似 IE 的前后浏览支持。EASTT 关系图提供 tooltips 支持。考虑到屏幕大小的限制，节点内文字可能不够清晰，为方便用户浏览，当鼠标移至节点上方时，一个包含该节点有关内容的透明窗口会及时出现；当鼠标移开后，这个透明窗口将自动消失。EASTT 关系图提供两种显示模式：对称型模式和树型模式。对称型模式中同层的节点将在屏幕范围内对称分布。树型模式中将每一个节点视为树根，在自己的控制范围内列出相关的下层节点。两种显示模式可自由切换。

(1) 类继承：显示类之间的继承状况，可以选择树型或对称型模式来显示。图 12-11 针对类继承情况展现出了两种不同的打开方式。

(2) 函数调用：显示函数调用情况。上层函数调用下层函数。它的打开方式也分为两种：树型和对称型。图 12-12 显示了函数调用的树型打开方式。

(3) 类/函数耦合：显示类/函数间的耦合情况，如图 12-13 所示。



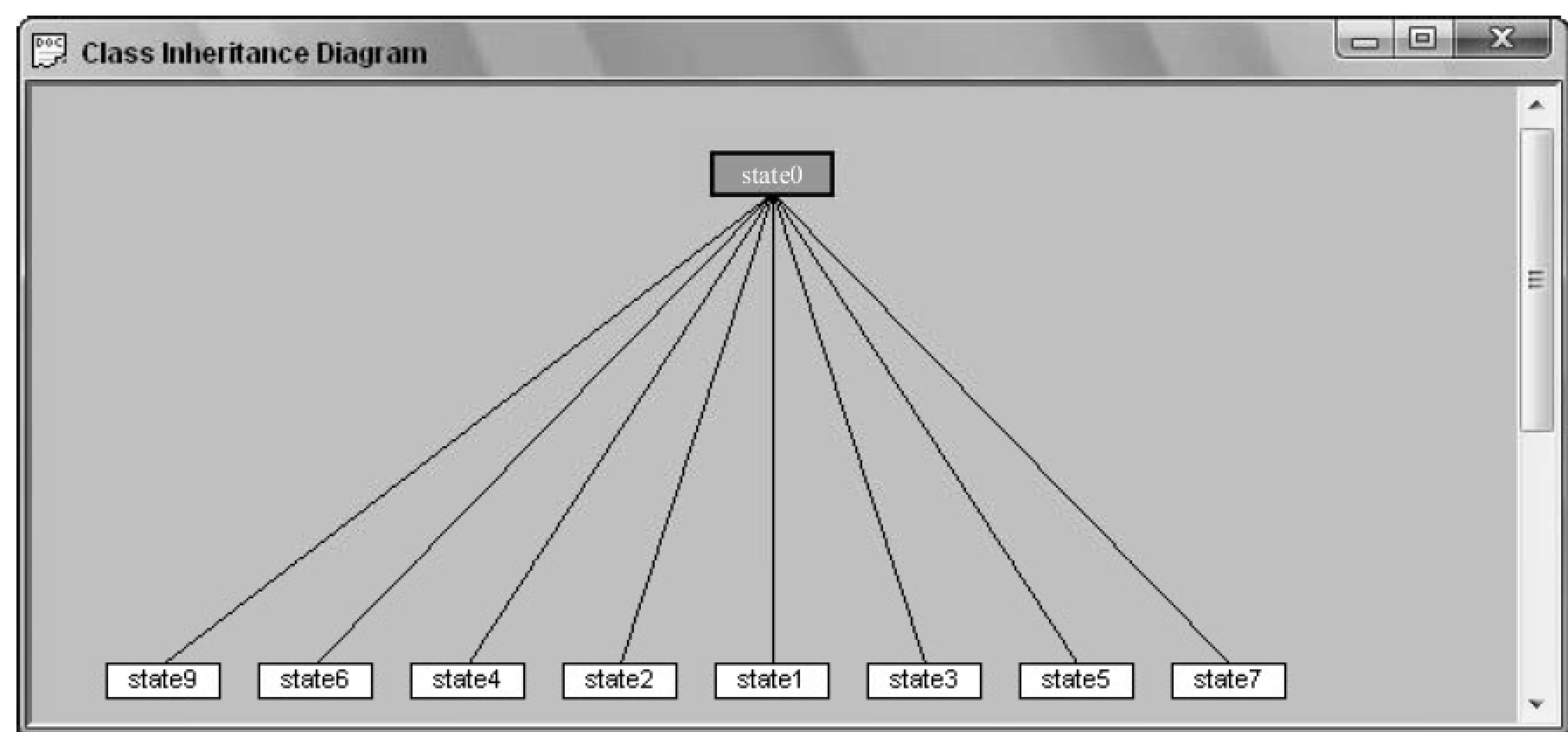


图 12-11　类继承的树型和对称型打开模式

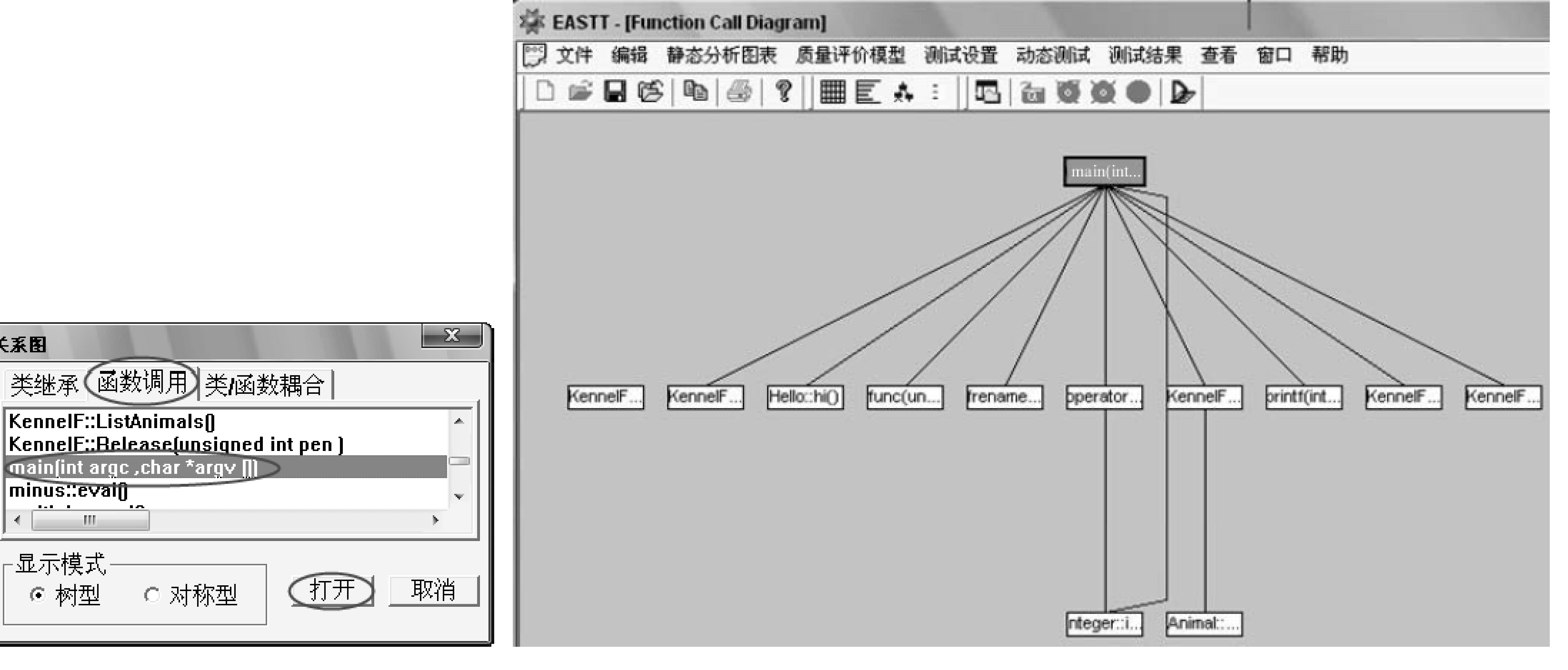


图 12-12　函数调用的树型打开方式

4) 层次流程图

层次流程图主要用于分析类或文件中的方法或者函数的层次结构。当用户选择“静态分析”|“层次流程图”菜单项时，EASTT 将打开如图 12-14 所示的对话框，可由用户通过类或文件两种方式选择要浏览的函数名，选中后单击“打开”按钮，工作区将产生一个分隔窗口。窗口左栏为一滚动窗口，用类似于 PAD 的方式表示该函数的结构；窗口右栏也为一滚动窗口，显示该函数对应的伪码。

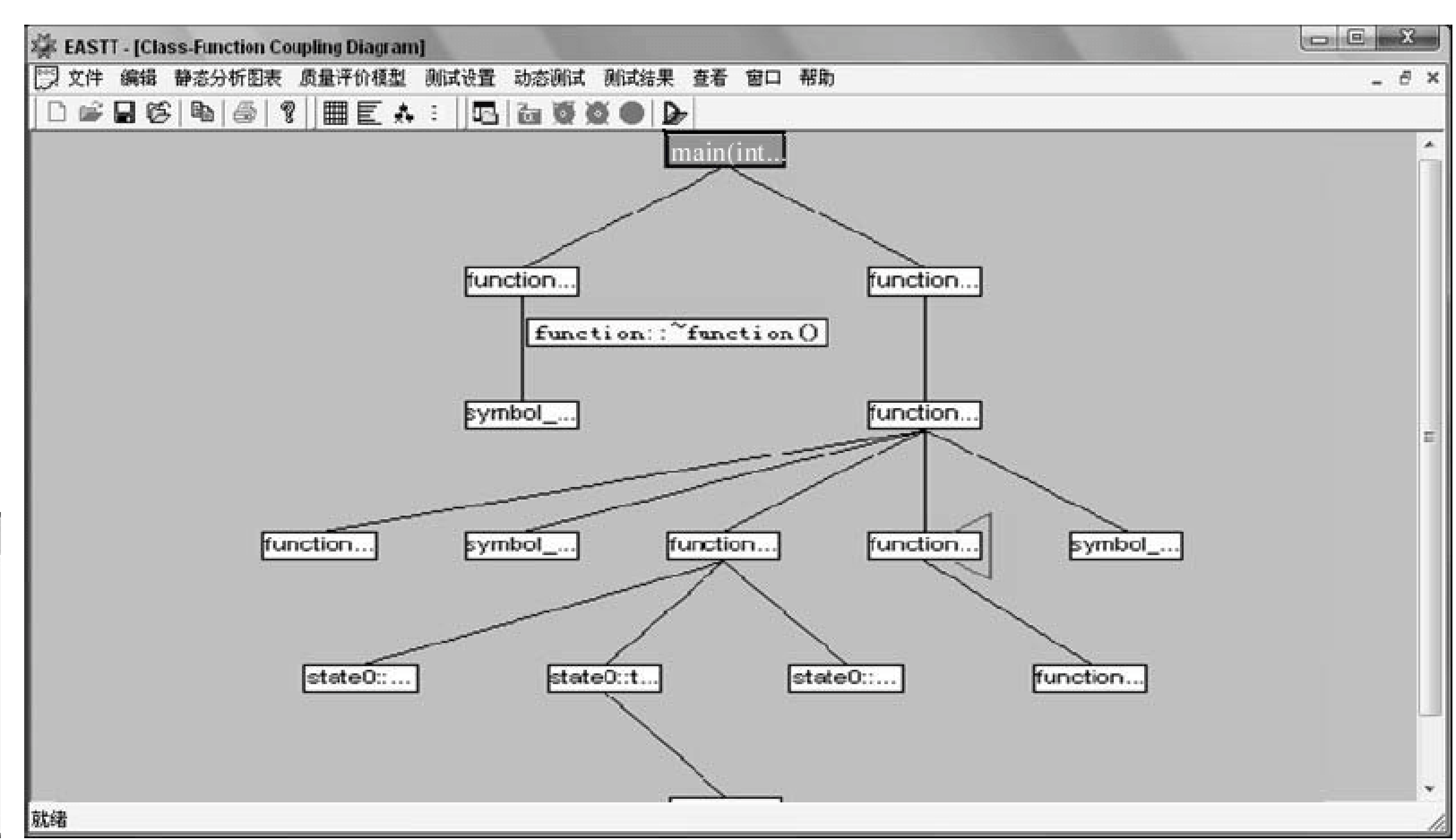


图 12-13　类/函数耦合调用图





图 12-14　层次流程图的函数选择

下面介绍 EASTT 层次流程图中的一些基本概念和功能。

(1) 基本概念

EASTT 对 ANSI C++的基本语法结构进行提取，给出了 Layer、Segment、Node 三个级别上的基本结构。不同级别的节点有不同的表示方法。Layer 型节点为可展节点，Segment、Node 型节点为不可展节点。Node 型的节点又可分为可收缩型和不可收缩型。Layer 型节点在展开后，它对应的同名 Node 型节点为可收缩型 Node 型节点，其他 Node 型节点均为不可收缩型 Node 型节点。

图 12-15 中的第一个 if 节点为 Layer 型可展节点，Visibl 节点为 Segment 型不可展节点，后面的 4 个节点均为 Node 型不可展节点。

后三个节点均为可收缩型 Node 型节点，func 节点为不可收缩型 Node 型节点。

(2) 功能

EASTT 层次流程图提供一种动态的层次性扩展/收缩功能。若用户希望进一步了解某个 Layer 型节点内部的具体情况，可双击该节点，EASTT 将自动展示出节点内部的具体结构。反之，当用户希望浏览某一段的整体结构时，可双击可收缩型 Node 型节点，EASTT

将自动隐藏节点内部的具体结构。

EASTT 层次流程图提供当前控制流显示功能。当用户在左栏单击某个节点时，该节点即为当前节点，函数控制流程中与当前节点相关的控制线都将变色。

EASTT 层次流程图支持节点与代码间的一一对应功能。当用户在左栏单击某个不可展节点时，右栏中该节点对应的代码行将变色；当用户在右栏单击某行代码时，EASTT 也将找到该代码行所对应的节点，使之变色。

EASTT 层次流程图提供节点查寻功能。能帮助用户理解某段代码在整个函数中的逻辑位置，而无须找遍函数的所有结构。当用户在右栏单击某行代码时，如果其对应节点为 Layer 型节点，用户可双击该节点，EASTT 将自动展示出该节点内部的第一层结构。如果有必要，用户可再次在右栏单击该行代码，左栏中对应节点一定在上述第一层结构中。若仍为 Layer 型节点，可重复以上步骤，直至找到某个 Node 型或 Segment 型节点为止。

EASTT 层次流程图提供显示自动调整功能。EASTT 能自动调整左右栏窗口，使变色的当前代码/节点显示在窗口的醒目位置。

EASTT 层次流程图提供 tooltips 支持。考虑到屏幕大小的限制，节点的文字内容可能不够清晰，为方便用户浏览，当鼠标移至节点上方时，一个包含该节点有关内容的透明窗口会及时出现；当鼠标移开后，这个透明窗口将自动消失。

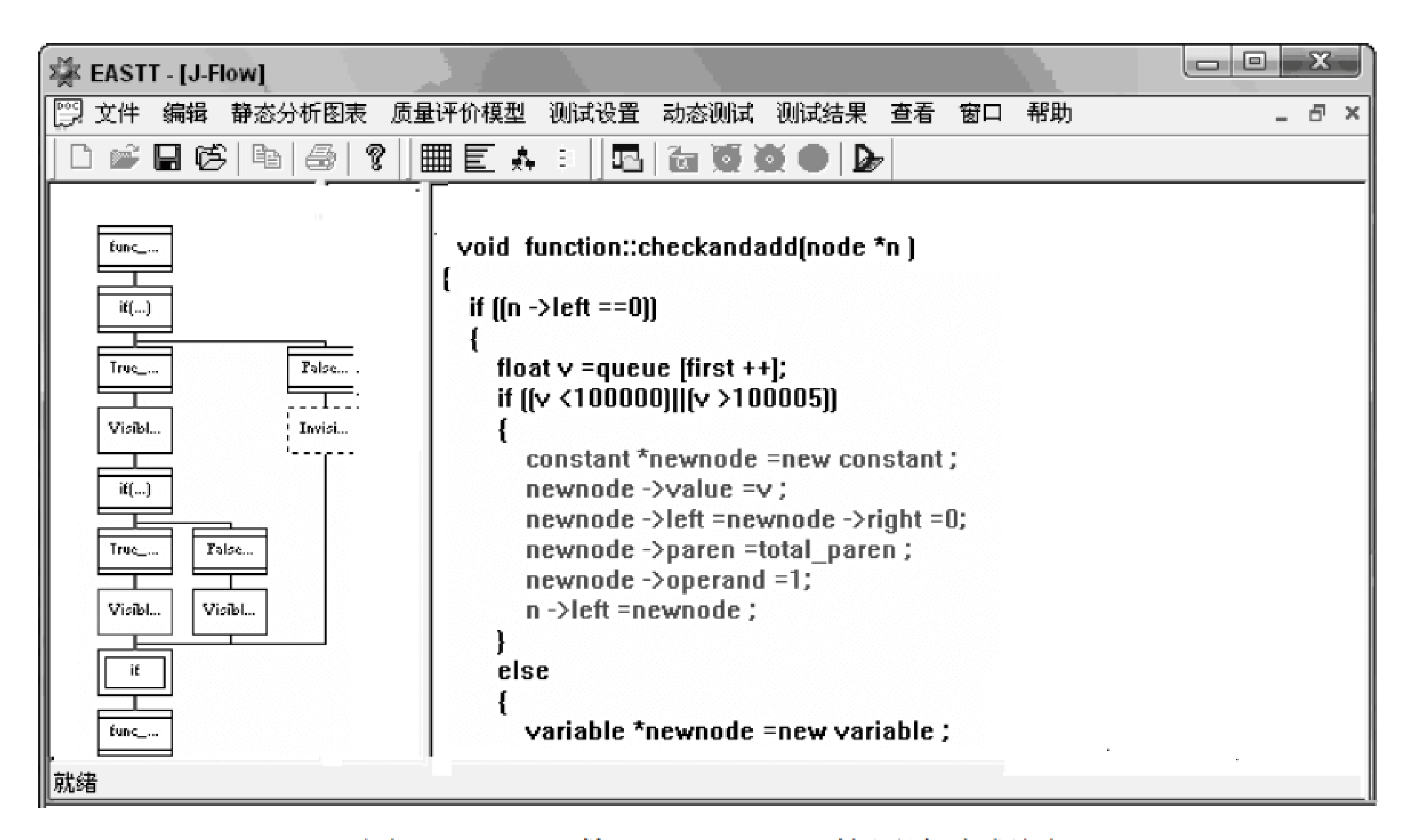


图 12-15　函数 checkandadd 的层次流程图

3. 三级质量模型

EASTT 软件支持符合 ISO 9126 标准的三级软件质量度量模型，用户可以根据需要使用内置的质量模型或自定义质量模型，来生成被评价软件的 Kiviat 图(雷达图)和饼状图，并用它对软件的质量进行评价。

三级质量模型包括度量元、质量准则、质量因素三个级别。度量元为最底层的级别，它直接提取程序在某一方面的信息，如函数语句数度量元。质量准则是高于度量元的一级，它通过对度量元进行组合运算来得到相应的数字值，从而实现对软件质量的定量分析。在 EASTT 软件的质量模型中，基本的质量准则有易于分析性(Analyzability)、易于测

试性(Testability)和稳定性(Stability)三个，用户还可以自定义新的质量准则。质量因素是最高级别的质量标准，同样，它通过对质量准则进行组合运算来得到数字值，基本的质量因素有可维护性(Maintainability)，用户可以自定义新的质量因素。

每个质量准则或质量因素都要制定一个评估等级，根据其量化的数值，确定软件在相应方面的质量等级，一般有 Excellent(优秀)、Good(良好)、Fair(一般)、Poor(差)4 个等级。

质量模型由质量模型编辑器、质量评价控制台、三级度量模型显示等部分组成。

(1) 质量评价控制台用于对被测试的部件进行选择，系统将针对被选中的部件进行测试，并返回测试结果。

(2) 由于软件质量评估对特定的应用程序有较强的依赖性，因而质量模型编辑器提供了对质量模型各级指标进行编辑的功能，用户可以根据应用系统的不同要求自行定义。同时为了方便起见，本系统也在参考了大量文献和标准的基础上设计了一套基本度量指标，作为系统默认指标以供用户直接使用。

(3) 三级度量模型显示采用 Kiviat 图和表格两种形式。

1) 质量评价控制台

选择“质量评价模型”|“质量评价控制台”菜单项，打开质量评价控制台。在质量评价控制台中选择被测试部件，系统将针对被选中的部件进行测试，并返回测试结果，如图 12-16 所示。



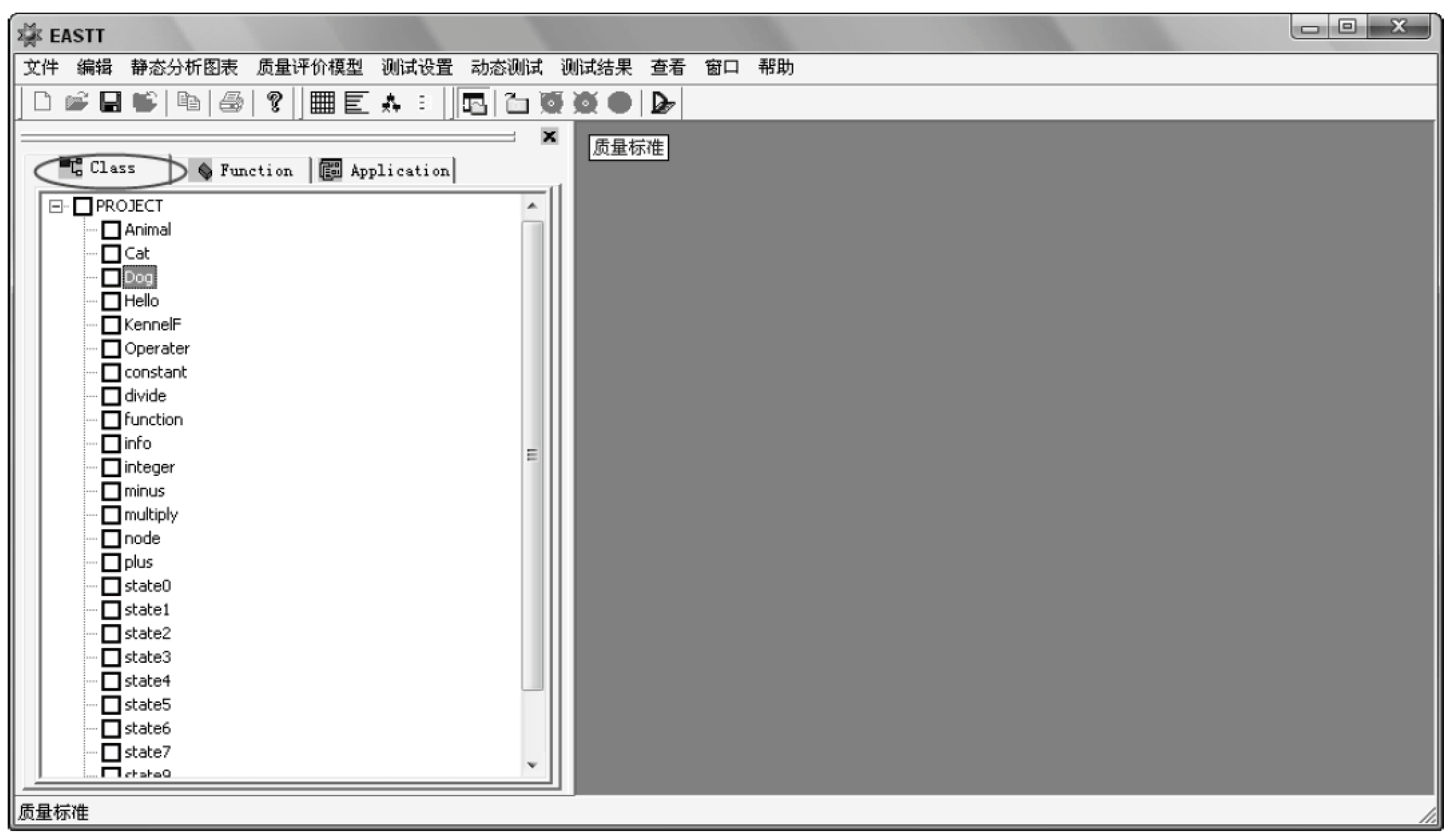


图 12-16　启动三级质量评价控制台

2) 质量模型指标定义文件(SQM)

系统提供了默认和用户自定义两种方式。默认 SQM 文件中定义了一套普遍适用的三级度量指标，使用该默认文件，可以不必再进行质量模型的自行编辑；用户也可以根据被测试系统的特点，使用质量模型编辑器自行定义一套符合应用需要的三级度量指标。

选择“质量评价模型”|“载入.SQM 文件”菜单项，即可导入/自定义 SQM 文件，如图 12-17 所示。

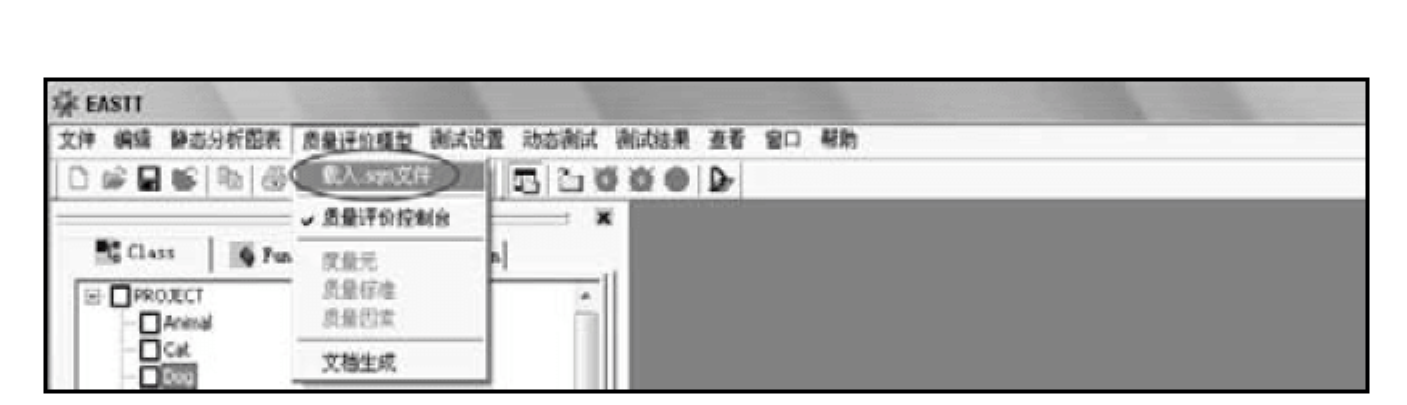




图 12-17　选择默认质量模型指标定义文件(SQM)

当用户在载入 SQM 文件对话框中选择“自定义 SQM 文件”单选按钮并单击 OK 按钮后，系统将打开质量模型编辑器。用户可以在编辑器中的提示操作下定义符合特定需要的三级度量指标。基本使用步骤如下。

对相应的作用域和度量级别编辑度量指标。其中作用域分为：类(Class)、方法(Function)和应用程序(Application)三个级别。各自分别有三级度量指标——度量元级、质量准则级和质量因素级。

度量元指标编辑：度量元指标的编辑可以从系统提供的基本度量元中选取；也可以通过组合若干个基本度量元来生成新的度量元指标。对每个度量元指标要通过给出最大值和最小值来限制其合理的取值范围。

质量准则指标编辑：质量准则指标由度量元指标组合而成，系统将根据其取值对照准则评估等级进行归类。所使用的质量准则指标可以从基本指标(Testability、Stability、Analyzability)中选取，也可以组合已有的度量元指标自行定义质量准则指标。每添加一个质量准则指标，就需要在等级评估设置中编辑等级评估信息。

质量因素指标编辑：质量因素指标由已有的质量准则指标组合而成，系统将根据其取值对照准则评估等级进行归类。所使用的质量因素指标可以从基本指标(Maintainability)中选取，也可以组合已有的质量准则指标自行定义新的质量因素指标。每添加一个质量因素指标，就需要在等级评估设置中编辑等级评估信息。

下面通过图 12-18～图 12-20 来说明如何自定义软件质量模型。







图 12-18　度量元指标编辑







图 12-19　质量准则指标编辑







图 12-20　质量因素指标编辑

3) 三级质量模型

(1) 基本度量元级：度量元是检验一个软件质量好坏的最基本元素，可以查看各度量元的名称和值。例如，NOC 为子类数目，也就是直接子类的数目，DOI 为继承树的深度，也就是类在继承树中的最大深度。操作时，选择“质量评价模型”|“度量元”菜单，打开该级窗口。在 Kiviat 图中，每个度量元指标用一条射线标识，被测试系统在相应指标上的取值通过落在射线上的圆点标出。根据指标取值范围的限定，实际值落在范围内的用绿色圆点标出，落在范围外的用红色圆点标出，如图 12-21 所示。

(2) 质量准则级：操作时，选择“质量评价模型”|“质量标准”菜单，打开该级窗口。在表格中，列出了被测试系统在相应质量准则指标上的等级取值。Kiviat 图中对每个质量准则都显示出了其组分中的各度量元名称及取值，如图 12-22 所示。

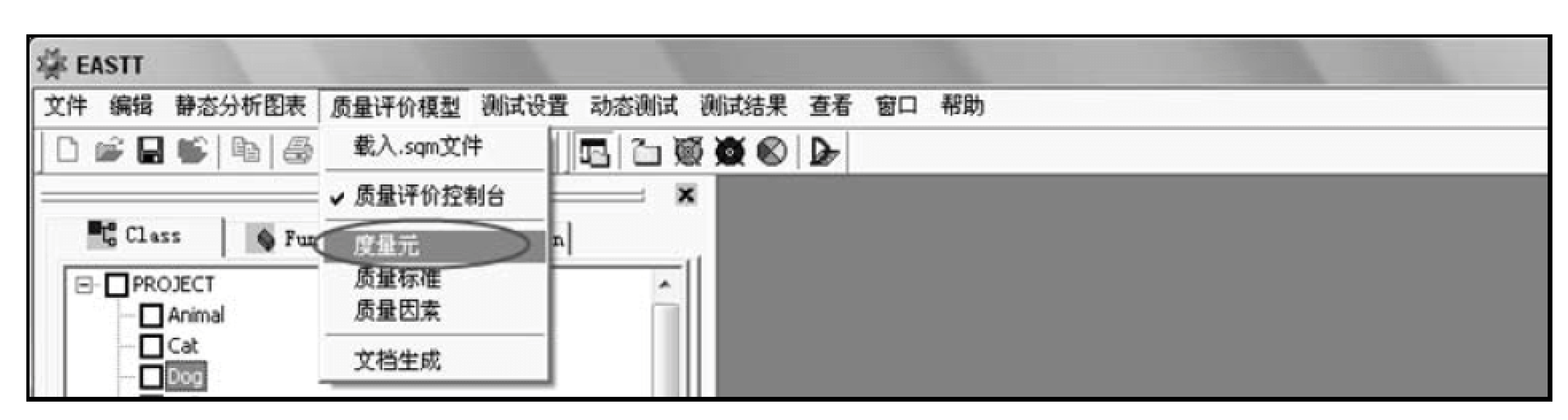


图 12-21　度量元取值的 Kiviat 图表示

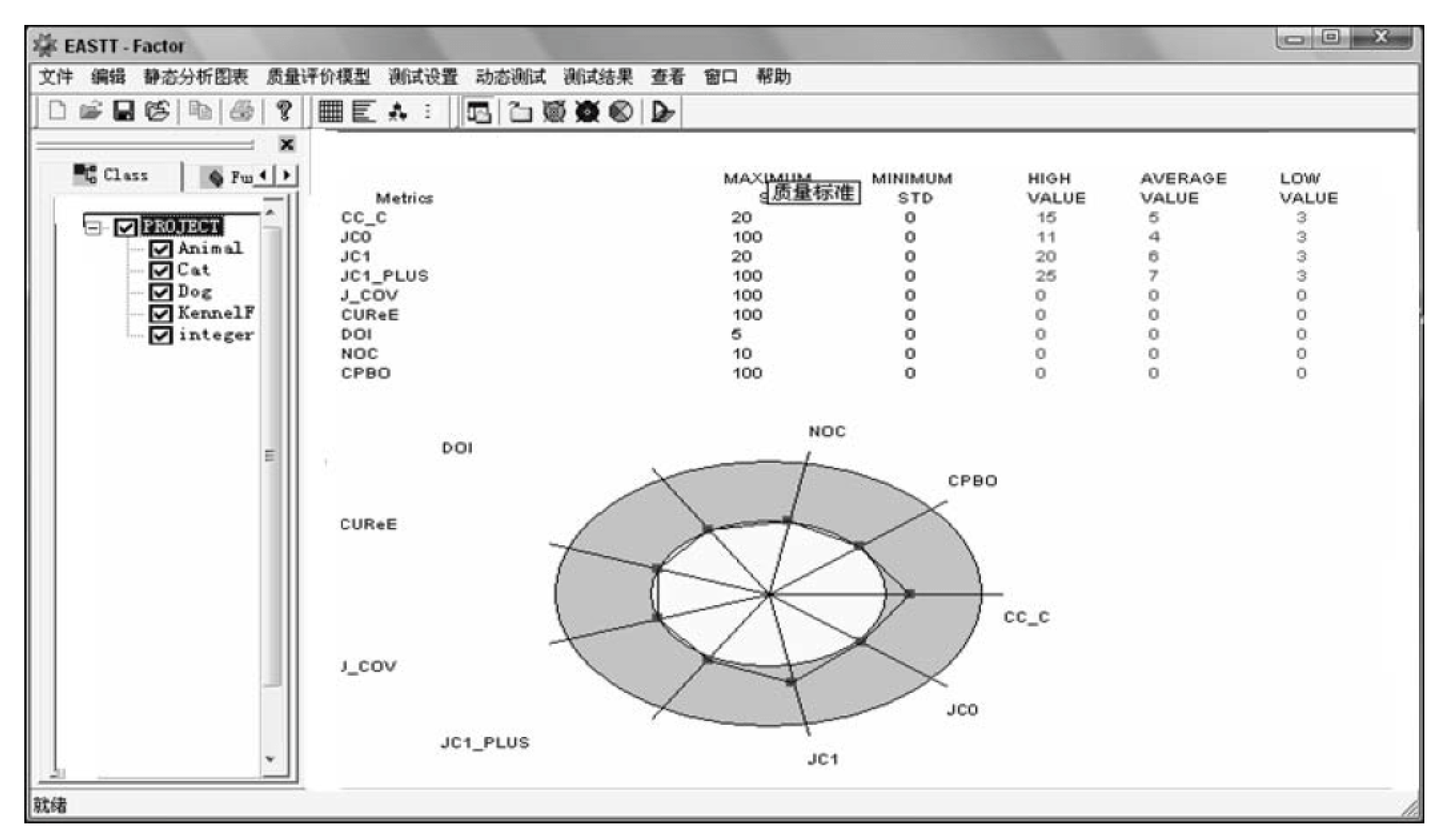


图 12-21　(续)



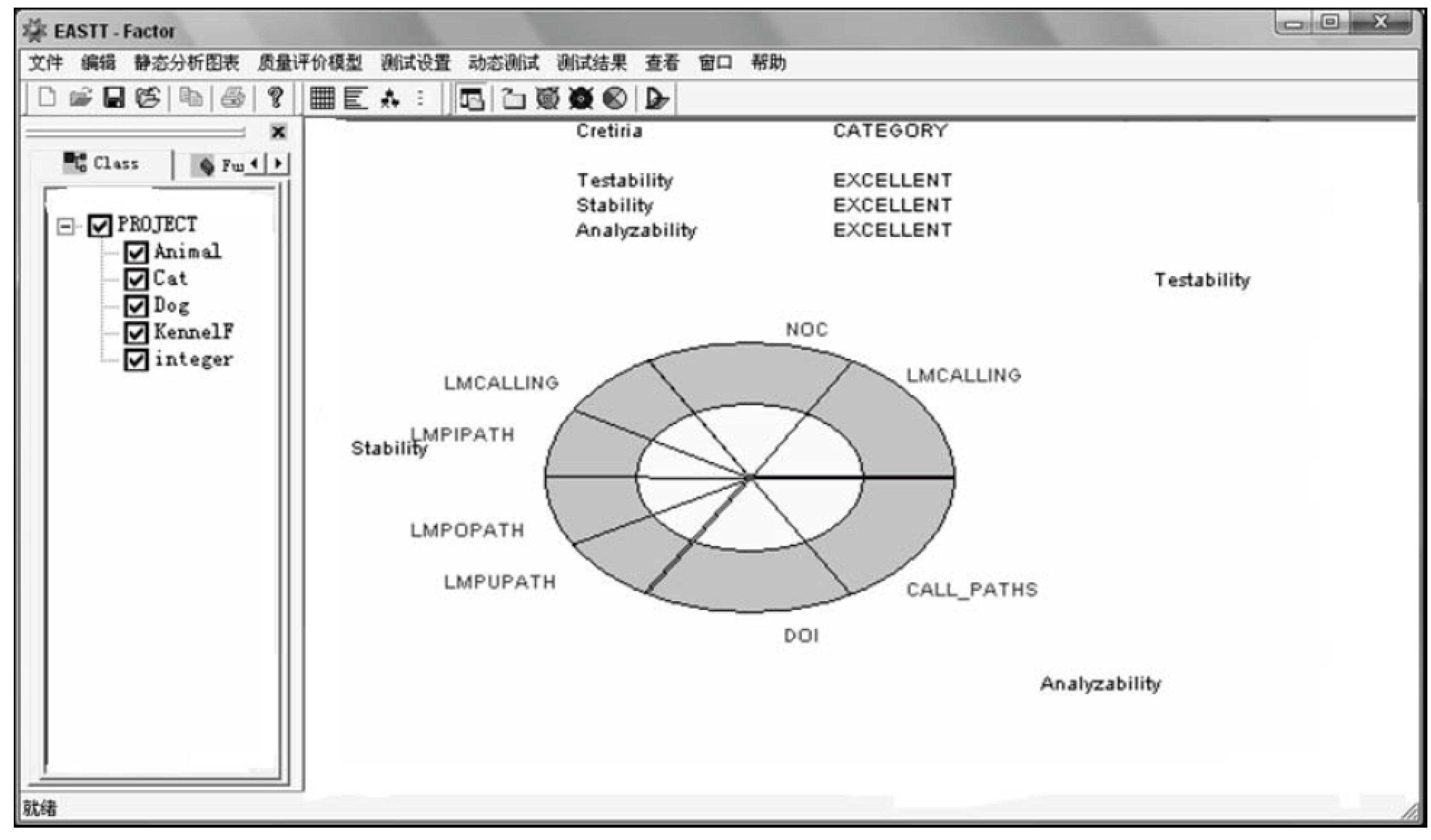


图 12-22　质量准则的 Kiviat 图表示

(3) 质量因素级：选择“质量评价模型”|“质量因素”菜单，打开该级窗口。该级别中，对落在质量因素相应分类中的部件(类、方法)的数目加以统计，各部分所占比例分别用表格和饼图的形式加以描述，如图 12-23 所示。



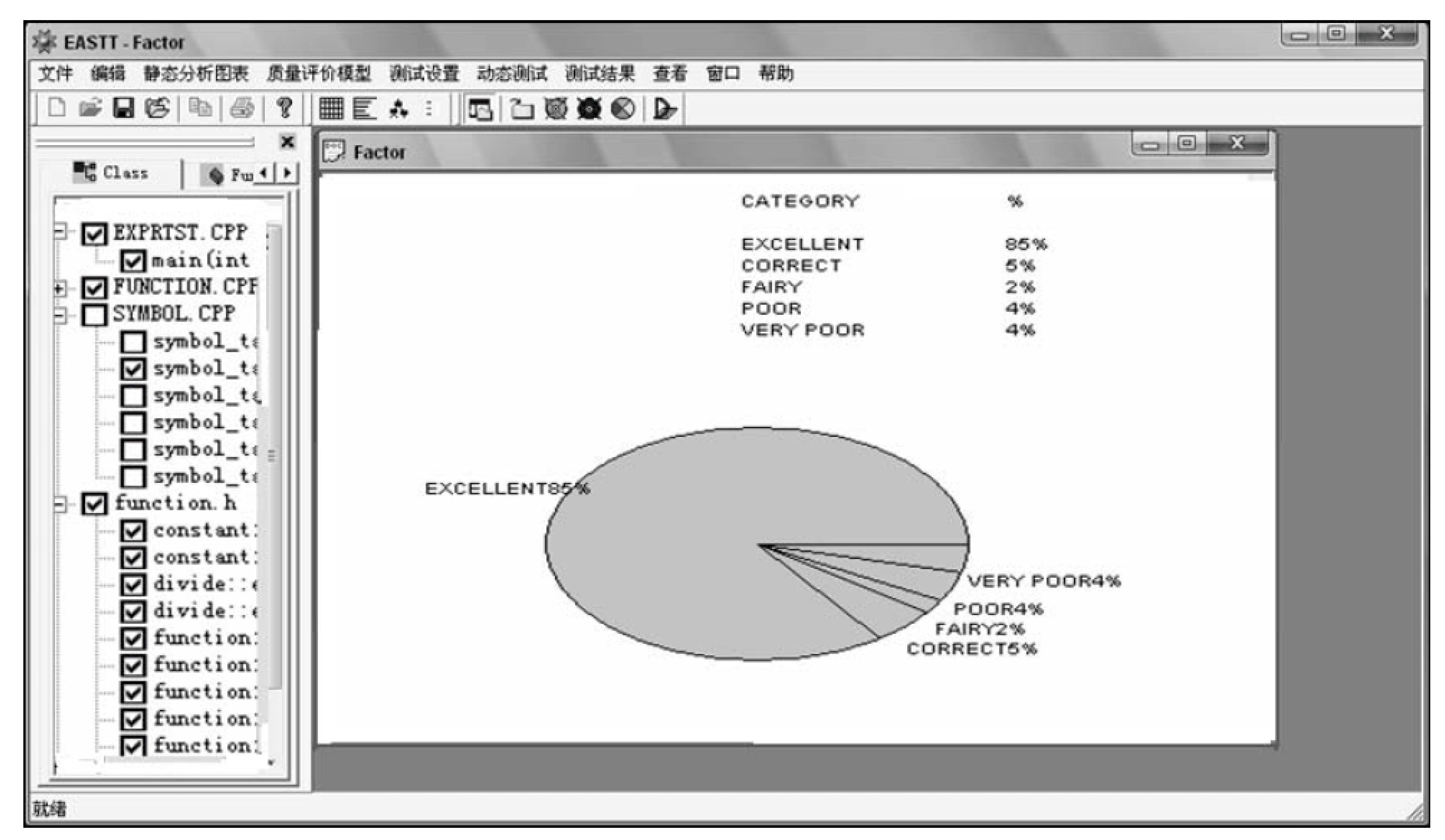


图 12-23　质量因素的表格和饼图表示

4. 动态测试运行

EASTT 支持两种测试方式：HostToHost 和 HostToTarget。HostToHost 方式中被测程序和测试工具在一台主机上运行；而 HostToTarget 中测试程序在主机上运行，被测程序在 PSOS 系统目标机上运行。可以根据项目文件的后缀名进行区分，.dsp 是 HostToHost 方式，.mak 则是 HostToTarget 方式。下面分别介绍这两种测试方式的测试过程。

1) HostToHost 测试方式

(1) 测试初始化在开始动态测试之前需要做一些初始化工作，初始化完毕之后便可以开始动态测试。

(2) 指定插桩文件和插桩粒度在“测试设置”菜单中选择需要进行插桩的文件和插桩粒度。共分为三个插桩粒度：Class、Function 和 Segment，如图 12-24 所示。





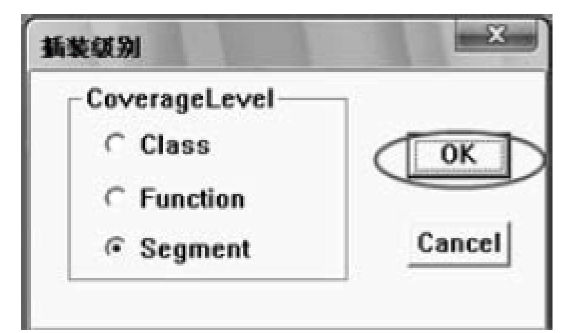


图 12-24　插桩策略选择

(3) 插桩。选择“动态测试”菜单中的“动态插桩”命令，根据选定的粒度对选定的文件进行插桩。若项目文件所在目录为 D，那么插桩所得到的文件在目录 D：\OOAtempuse\Target 下；如果被测程序已经插桩，并已编译得到可执行程序了，那么再次进行测试时就不需要进行插桩，在添加测试用例时直接指定该程序即可。

(4) 编译插桩得到的文件。将 EASTT 安装目录下“gamma 通信库”文件夹中的 HosttoHost.cpp 和 probe.h 文件复制到被测项目文件夹下的 OOATempuse\target 目录下。该目录中的程序文件是经过插桩后的程序文件，用于进行动态测试分析。复制完成后，用 Visual C++ 6.0 将 OOATempuse\ target 目录下的程序文件和 HosttoHost.cpp、probe.h 文件添加到一个项目中，然后选择 VC 菜单栏上的 Project(项目)|Settings(设置)命令。在对话框右边选择 C/C++选项卡，在 Category(类别)下拉列表中选择 Code Generation；然后在 Use run-time library(运行时链接库)框中选择 Debug Multithreaded，使用多线程方式；最后编译连接，生成可执行文件，即可完成测试设置。当被测程序再次进行动态测试分析时，可跳过测试设置这一步。

(5) 添加测试用例。选择“编辑”|“新建 Tcase”命令，创建一个测试用例。在打开的对话框中，Command 指插桩后编译连接得到的可执行文件的名称，Work Dir 是该文件所在的目录，Parameter 是执行该文件时的命令行参数，Description 是对该测试用例的相关描述，如图 12-25 所示。

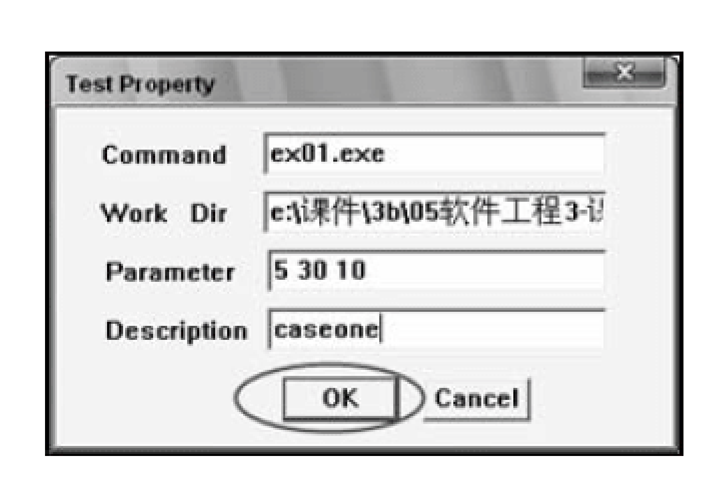


图 12-25　新建测试用例

(6) 执行测试。根据创建该测试用例时输入的路径、命令和参数信息，以命令行的方式调用插桩后得到的可执行文件进行测试。注意确保插桩后的可执行文件的路径和命令名的正确性；也可以单击选中某个测试用例，右击，从弹出的菜单中选择 Run 命令即可运行该测试用例，如图 12-26 所示。程序运行完毕后，对应的测试用例图标就会改变成另一种紫色的图标，表明该测试用例已经运行过，蓝色图案则表示该测试用例未被测试。

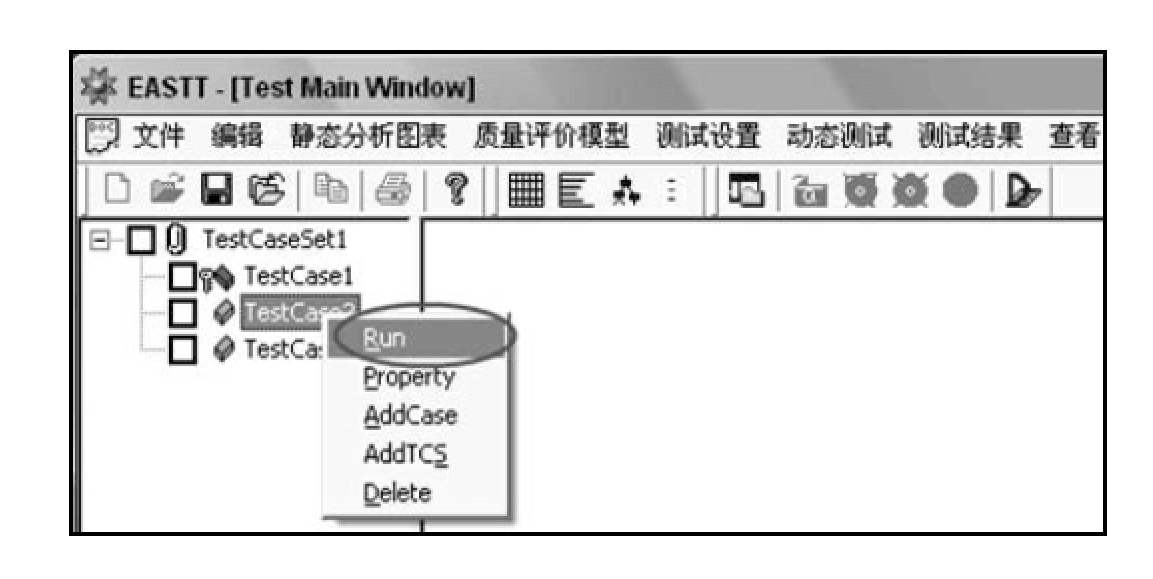


图 12-26　测试用例是否执行显示

(7) 查看测试结果。测试结果主要有：测试覆盖、动态执行、测试用例最小化、测试效率、动态 PPP、动态层次流程图和测试开销等分析。

2) HostToTarget 测试方式

HostToTarget 方式和 HostToHost 方式大致相同。其主要过程如下。

(1) 测试初始化。在开始动态测试之前也需要做一些初始化工作，初始化完毕之后便可以开始动态测试。

(2) 指定插桩文件和插桩粒度。在“测试设置”菜单中选择需要进行插桩的文件和插桩粒度。共分为三个插桩粒度：Class、Function 和 Segment。

(3) 插桩。选择“动态测试”菜单中的“动态插桩”命令，根据选定的粒度对选定的文件进行插桩。假设项目文件所在目录为 D，那么插桩所得到的文件在目录 D：\OOAtempuse\Target 下。如果被测程序已经插桩，并已编译得到可执行程序了，那么再次进行测试时就不需要进行插桩，直接在目标机上启动该可执行程序即可。

(4) 设置主机 IP 和通信端口。在测试设置中有 HostToTarget 设置选项，用于设置测试工具所在主机的 IP 和通信端口。使用该 IP 和端口同目标机(运行 PSOS 被测程序的机器)进行通信。从目标机收集相应的测试信息。

(5) 编译插桩得到的文件。利用 PSOS 的开发工具 PRISM+，把上述目录下的文件和测试工具生成的 setting.h 文件，以及 HostToTarget 库文件 HosttoTarget.cpp 和 libprobe.h 添加到一个项目中，编译连接，产生可执行文件。

(6) 添加测试用例。选择“编辑”|“新建 Tcase”命令，创建一个测试用例。

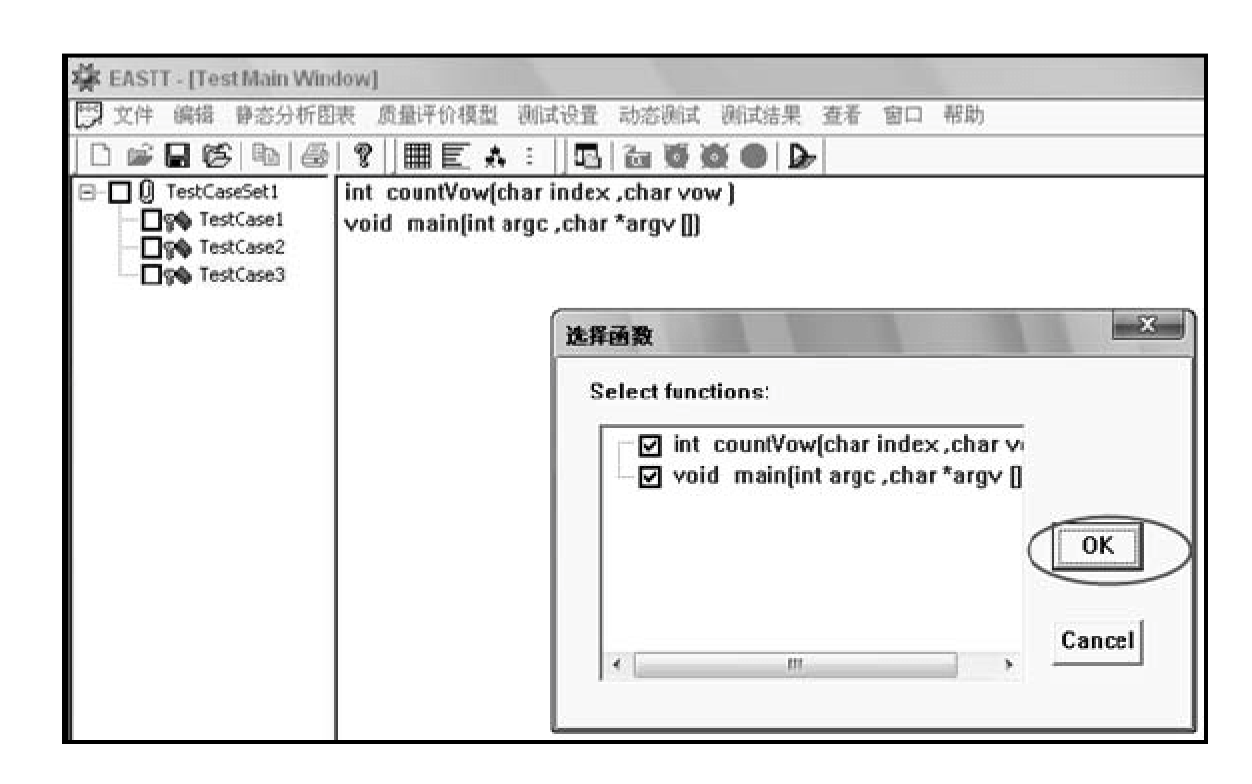
(7) 开始测试。HostToTarget 的测试和 HostToHost 的测试不大一样，因为被测程序运行在另外一台机器上。选择“动态测试”中的“开始测试”命令收集测试数据，然后在目标机上启动步骤(5)得到的可执行程序。注意：确保可执行程序在目标机上开始运行之前，主机测试工具已经开始测试，否则会丢失测试信息。在被测程序在目标机上执行完毕后，请稍等片刻，然后选择“动态测试”|“结束测试”命令结束本次测试过程，这样可以保证所有的测试信息都被收集到而没有被丢失。

(8) 查看测试结果。该步骤同 HostToHost 测试方式相同。

这里需要注意的是：PSOS 已经退出历史舞台，有兴趣的读者可将 EASTT 移植到 VxWork 上。

5. 覆盖测试分析

在“测试设置”|“覆盖分析”级别中进行选择，共有：File、Class、Function 和 Segment 4 个级别。选择完级别后，测试控制主窗口中右边便会根据左边窗口中高亮的测试用例进行相应的覆盖显示。测试用例前面的复选框被用于显示统计结果时对测试用例进行选择，如图 12-27 所示。



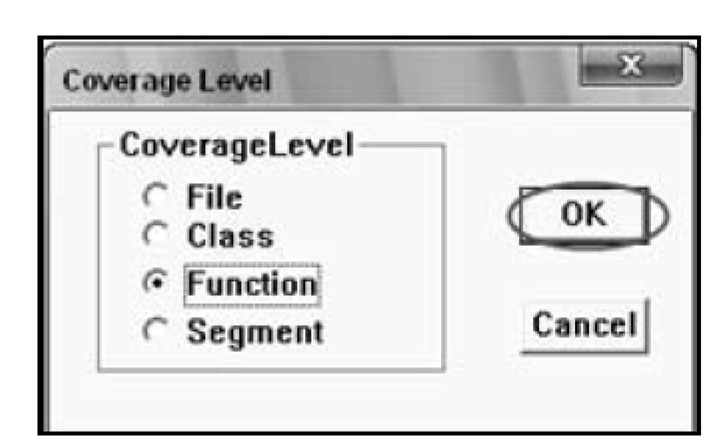


图 12-27　测试级别及测试用例的选择

1) 覆盖分析的结果直接显示在测试控制台中

选中某个测试用例，此测试用例所覆盖到的程序元素(文件、类、函数、代码段)在窗口右栏中就将以蓝色字符显示，未覆盖到的程序元素则以黑色字符显示，如图 12-28 所示。

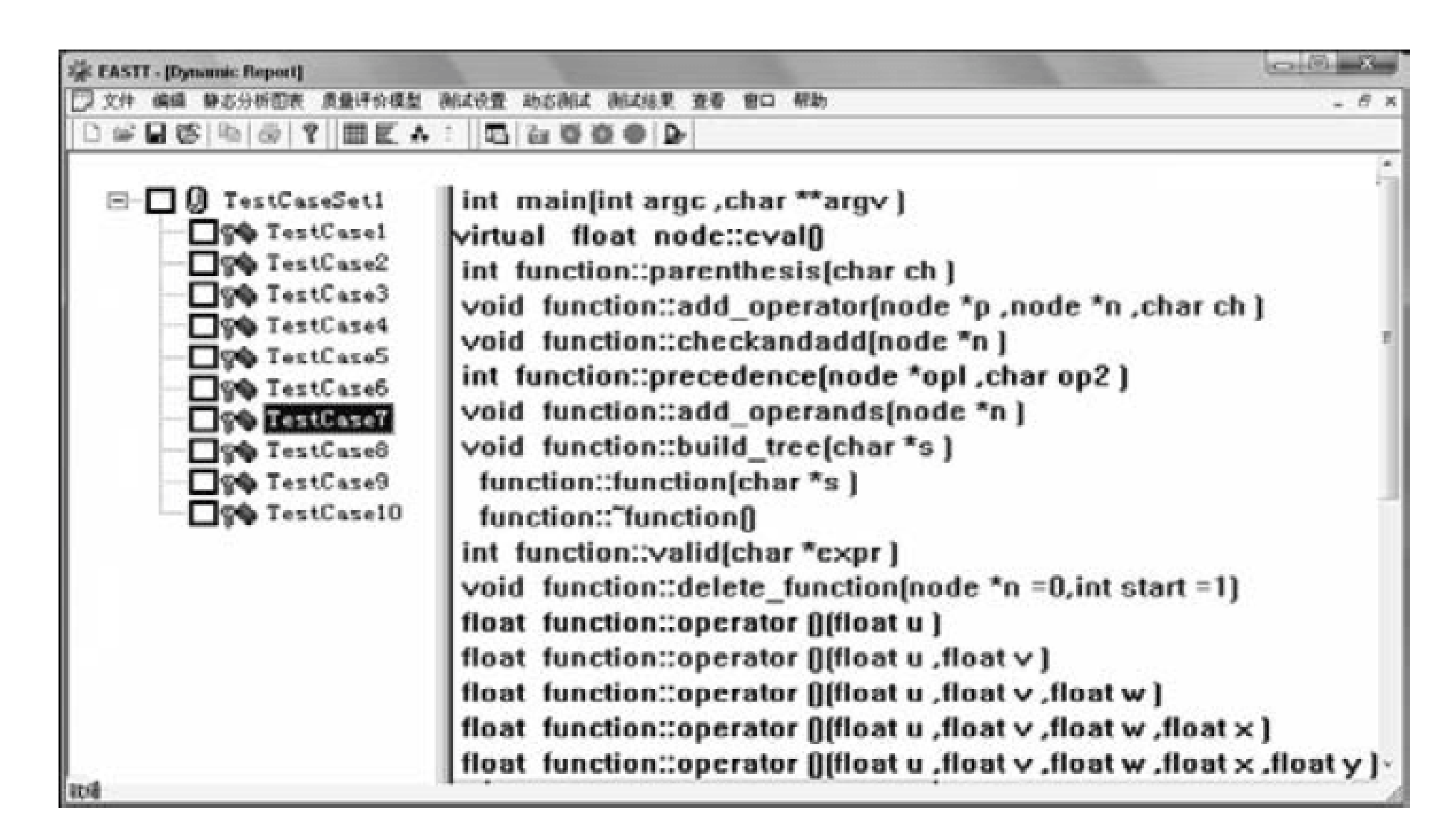


图 12-28　测试执行结果直接显示在控制台中

2) 测试结果以表格的形式显示

对图 12-29 中左边复选框中打“√”的测试用例集合进行统计，在表格中进行显示。窗口中包含 30 个动态测试分析表格，单击右侧的红色字母即可查看对应表格，如图 12-29 所示。

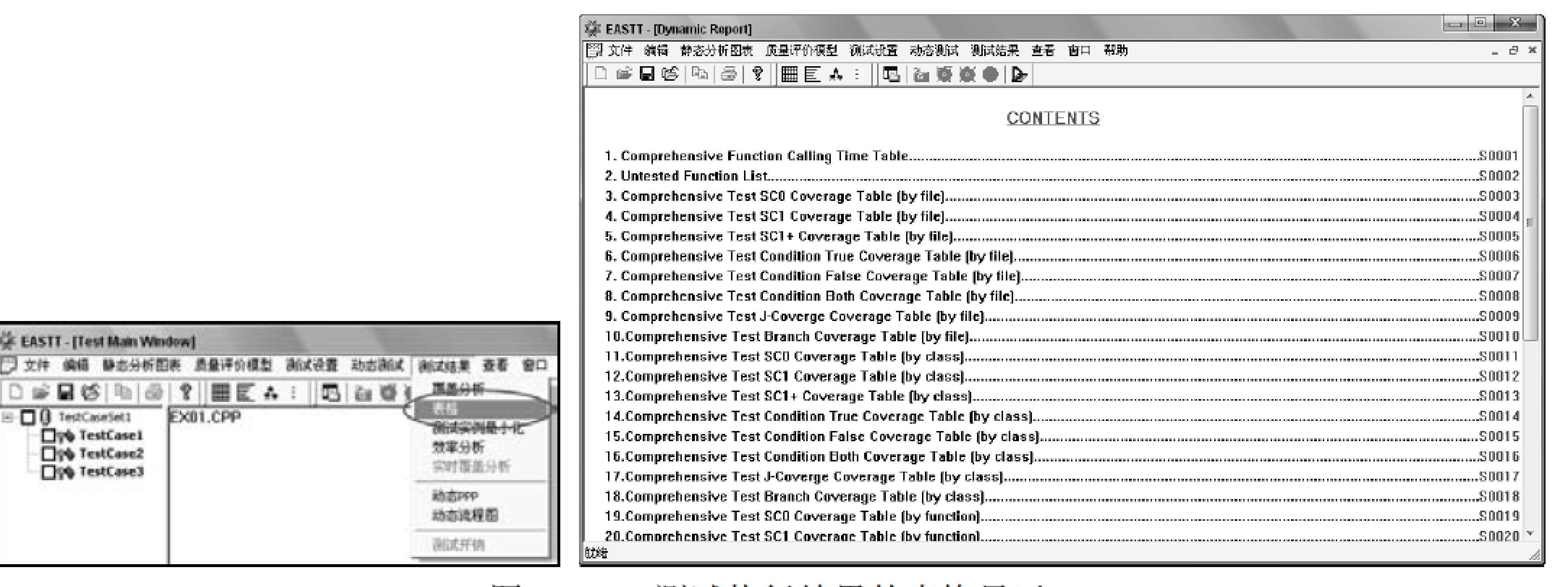
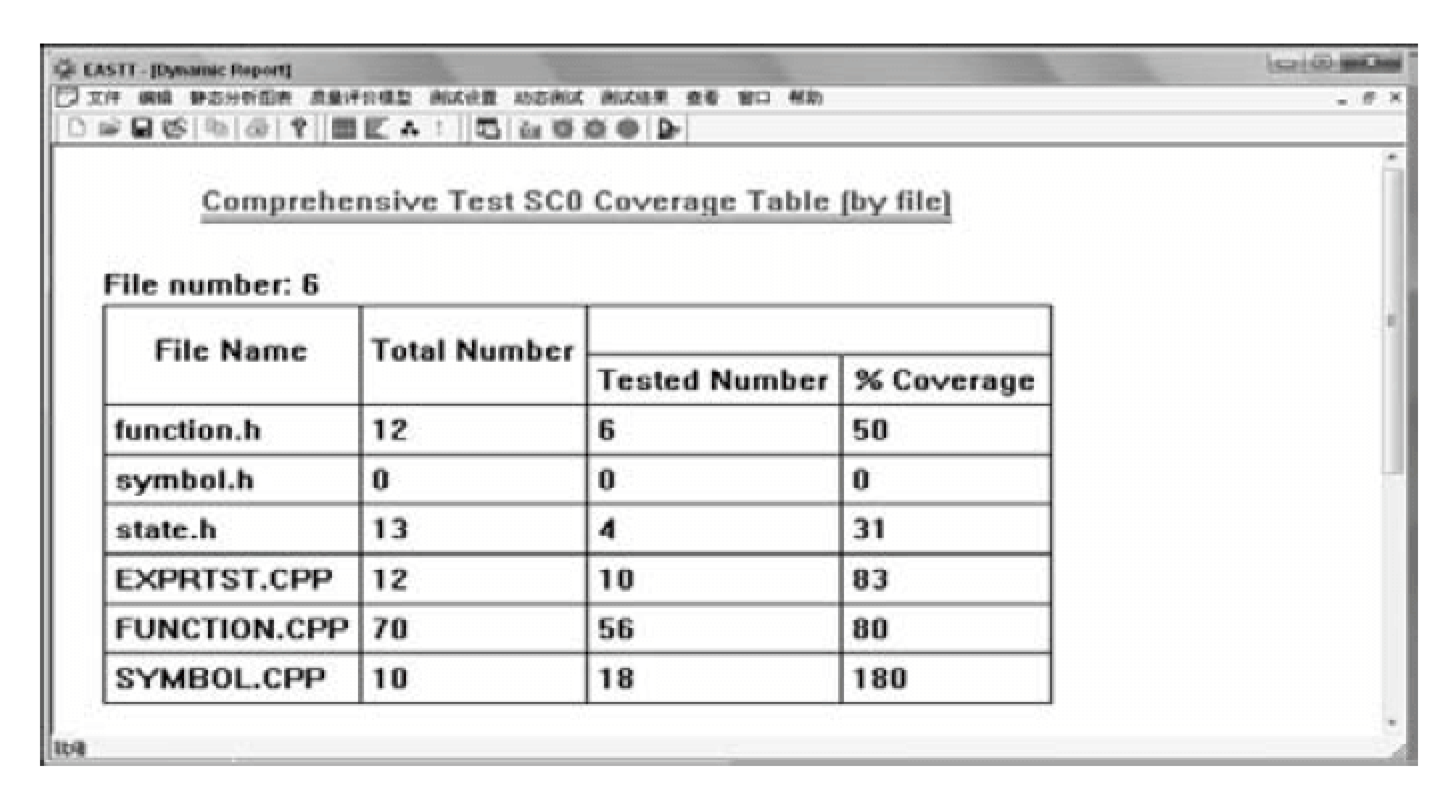


图 12-29　测试执行结果的表格显示



Comprehensive Test SC0 Coverage Table [by file]

File number: 6

File Name	Total Number		
		Tested Number	% Coverage
function.h	12	6	50
symbol.h	0	0	0
state.h	13	4	31
EXPRTST.CPP	12	10	83
FUNCTION.CPP	70	56	80
SYMBOL.CPP	10	18	180

图 12-29　(续)

表格中包括如下 30 个动态测试分析表格：

comprehensive function calling time table

untested function list

comprehensive test sc0 coverage table (by file)

comprehensive test sc1 coverage table (by file)

comprehensive test sc1+ coverage table (by file)

comprehensive test condition true coverage table (by file)

comprehensive test condition false coverage table (by file)

comprehensive test condition both coverage table (by file)

comprehensive test j-coverage coverage table (by file)

comprehensive test branch coverage table (by file)

comprehensive test sc0 coverage table (by class)

comprehensive test sc1 coverage table (by class)

comprehensive test sc1+ coverage table (by class)

comprehensive test condition true coverage table (by class)

comprehensive test condition false coverage table (by class)

comprehensive test condition both coverage table (by class)

comprehensive test j-coverage coverage table (by class)

comprehensive test branch coverage table (by class)

comprehensive test sc0 coverage table (by function)

comprehensive test sc1 coverage table (by function)

comprehensive test sc1+ coverage table (by function)

comprehensive test condition true coverage table (by function)

comprehensive test condition false coverage table (by function)

comprehensive test condition both coverage table (by function)

comprehensive test j-coverage coverage table (by function)

comprehensive test branch coverage table (by function)

comprehensive class test coverage table (by file)

comprehensive function test coverage table (by file)

comprehensive function test coverage table (by class)

comprehensive test coverage table (by project)

3) EASTT 测试用例最小化分析

在菜单栏选择“测试结果”|“测试用例最小化”命令，打开“测试用例最小化”对话框。在对话框左栏选中要进行最小化的测试用例，在中间选择最小化级别，然后单击 Minimum 按钮，软件将对所选测试用例进行最小化，并将所需的最小用例集显示在窗口右框中，如图 12-30 所示。单击 Clear 按钮将清空最小化结果，然后就可以选择其他测试用例进行用例最小化了。

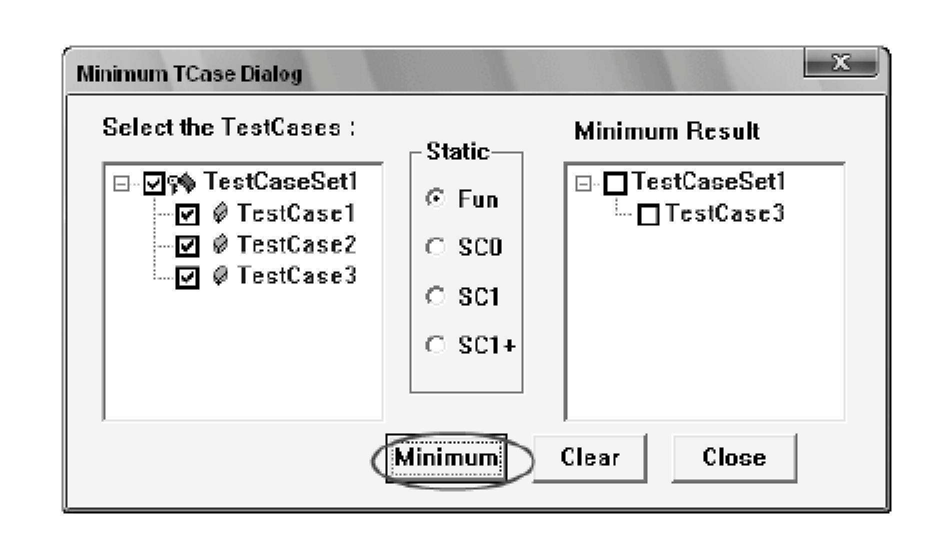


图 12-30　测试用例最小化

4) EASTT 测试效率分析

在测试控制台中勾选一些测试用例，然后在菜单栏上选择“测试结果”|“效率分析”命令，打开效率分析窗口，上面显示了所选测试用例的测试效率表。效率分析有 4 个级别：Function、Sc0、Sc1 和 Sc1+。可以在“测试设置”|“效率分析”级别中进行选择。效率分析结果如图 12-31 所示。

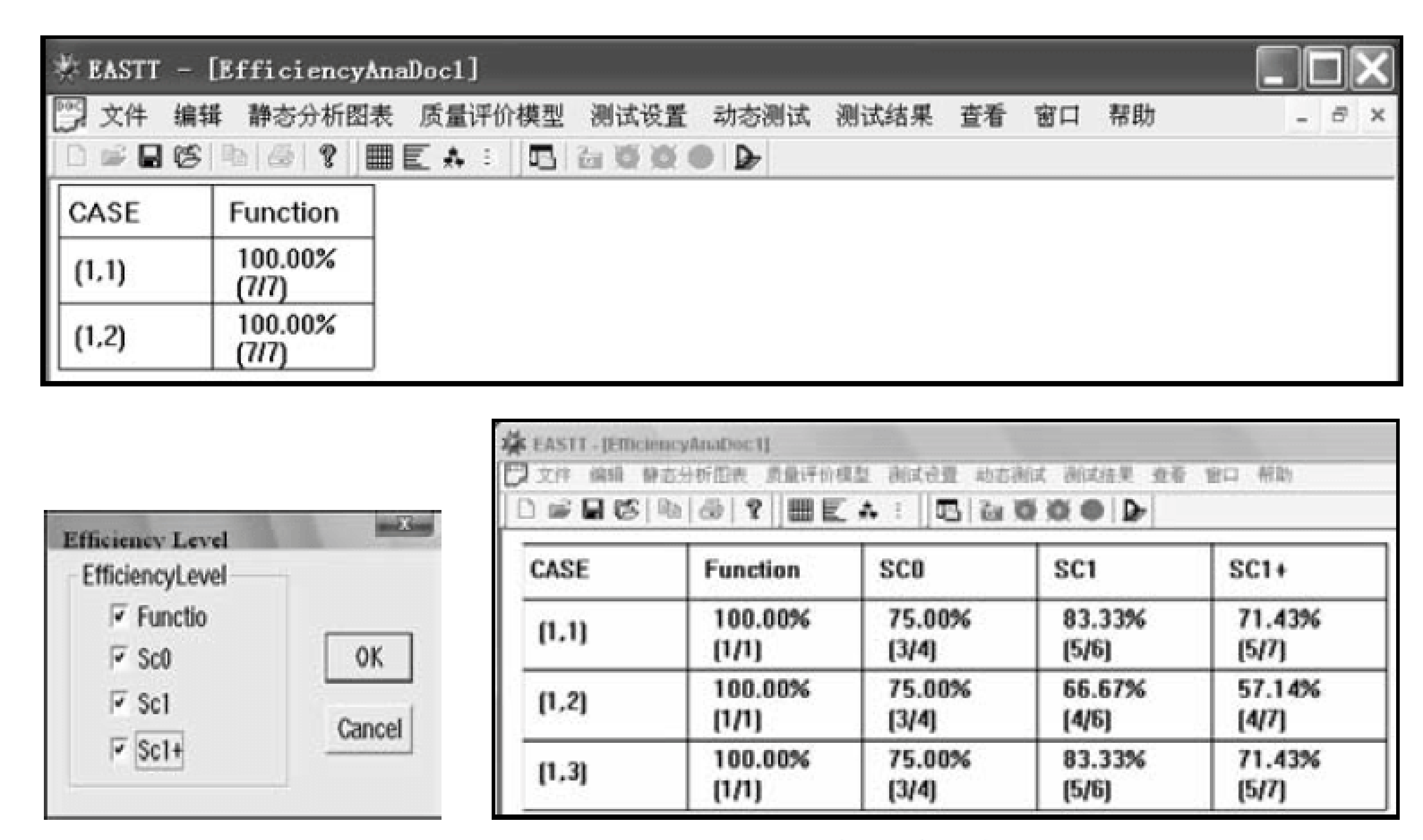


图 12-31　测试用例效率分析结果

5) EEAST 动态 PPP(动态关系图)分析

EASTT 动态 PPP 主要分析被测项目中各函数间的调用关系，并给出了相应的量化调用图。

当用户选择“测试结果”|“动态 PPP”菜单项时，EASTT 将打开一对话框，用户可在其中选择浏览函数，选中后单击“打开”按钮，工作区将产生一个滚动窗口，图形化地显示选中函数相应的调用关系图。动态 PPP 图的使用方法同结构分析中的关系图类似，不同之处在于表示调用关系的连线上会显示所选测试用例对这些调用关系的使用次数，如图 12-32 所示。

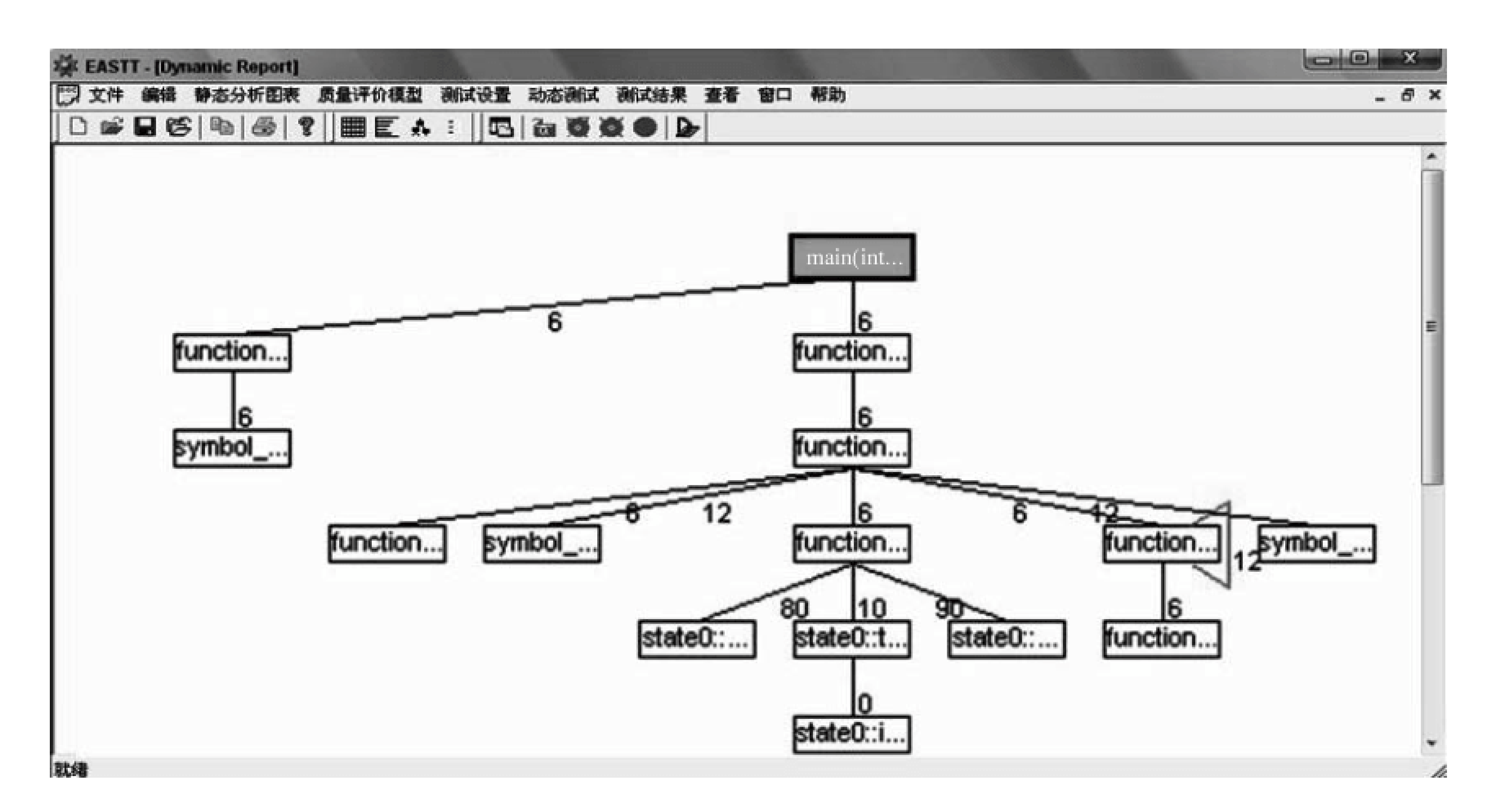


图 12-32　动态 PPP 图(动态关系图)

6) EASTT 动态层次流程图分析

当用户选择“测试结果”|“层次流程图”菜单项时，EASTT 将打开一对话框，在其中用户可通过类或文件两种方式之一来选择要浏览的函数名，选中后单击“打开”按钮，工作区将产生一个分隔窗口。窗口左栏为一滚动窗口，用类似问题分析图(PAD)的方式表示该函数的结构；窗口右栏也为一滚动窗口，显示该函数所对应的伪码。

动态层次流程图的基本概念、功能及使用可参见 EASTT 静态分析之层次流程图。每个节点的右上方均给出一整数值，该值表示当前测试用例覆盖该节点的次数，如图 12-33 所示。

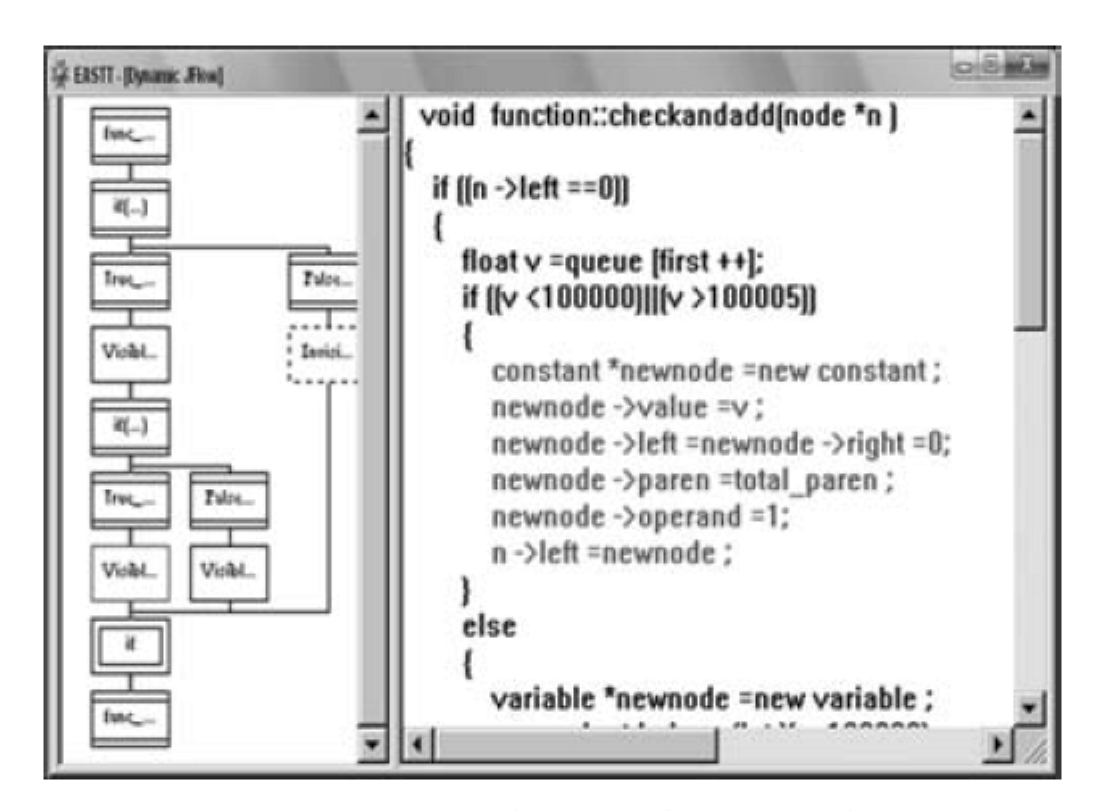


图 12-33　动态层次流程图分析

6. EASTT 测试开销分析

对每个测试用例的一次执行进行估计，看看测试的开销有多大，最后结果以插桩代码的运行时间占被测程序总运行时间的百分比形式给出，如图 12-34 所示。



图 12-34　测试开销分析——插桩代码的运行时间占被测程序总运行时间的比例

7. EASTT 测试分析文档生成器

EASTT 测试分析文档生成器是指将测试的结果进行汇总并生成 HTML 格式的文档，其内容包括被测文件列表、静态分析结果表格、动态测试覆盖表格等，如图 12-35 所示。每打开一个项目，项目目录中就会有一个相应的 HTML 文件自动生成，用户可以在任何阶段对其进行显示或打印。

需要指出的是：文档中的动态测试数据结果部分只在系统进行动态测试之后才会生成；未进行动态测试时，文档中将只包含静态分析部分的数据结果。

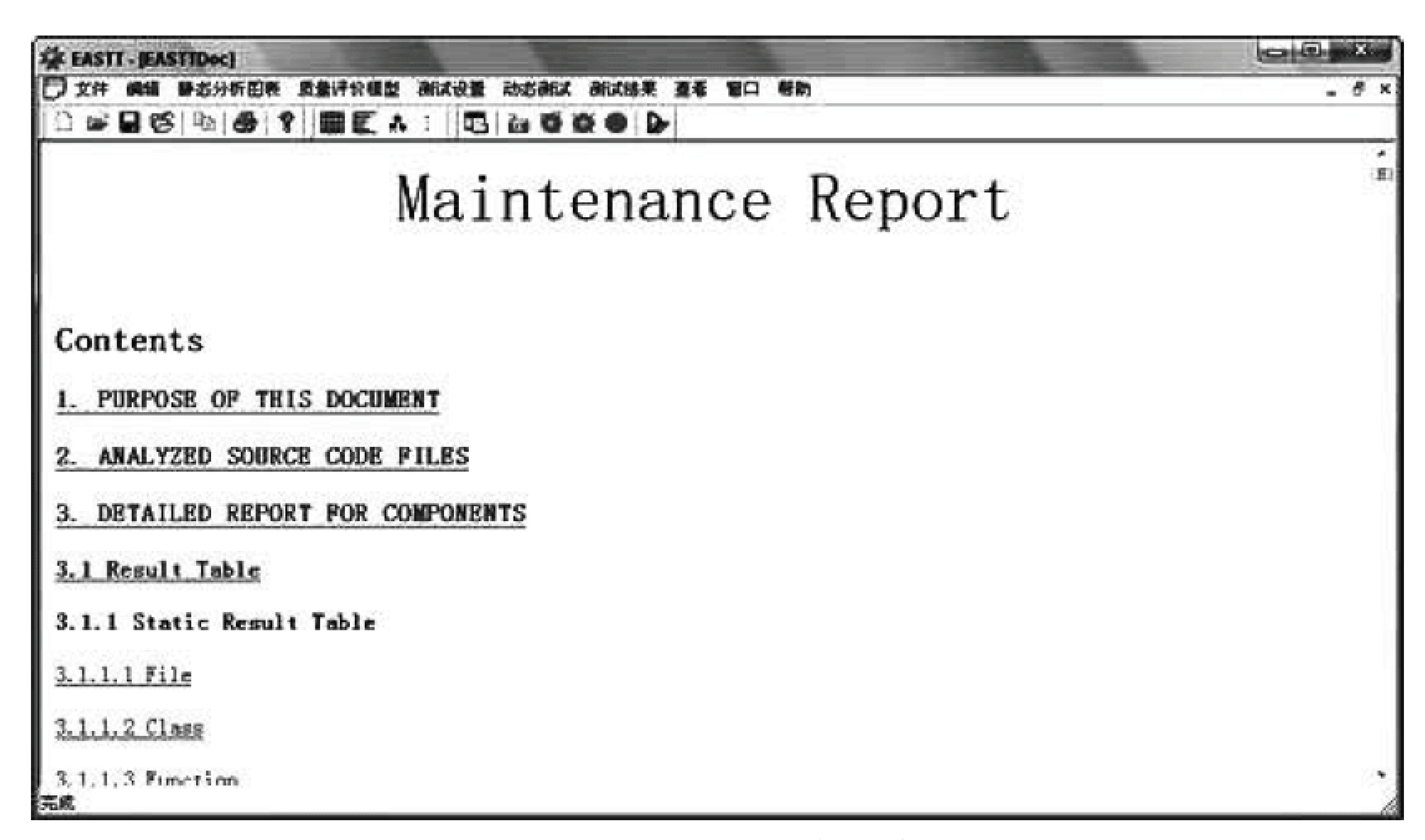


图 12-35　测试分析文档

12.4　EASTT 评测工具具体使用举例

使用例子为：对 EASTT\sample\example02\ex02.cpp 加以更改扩展的例子。

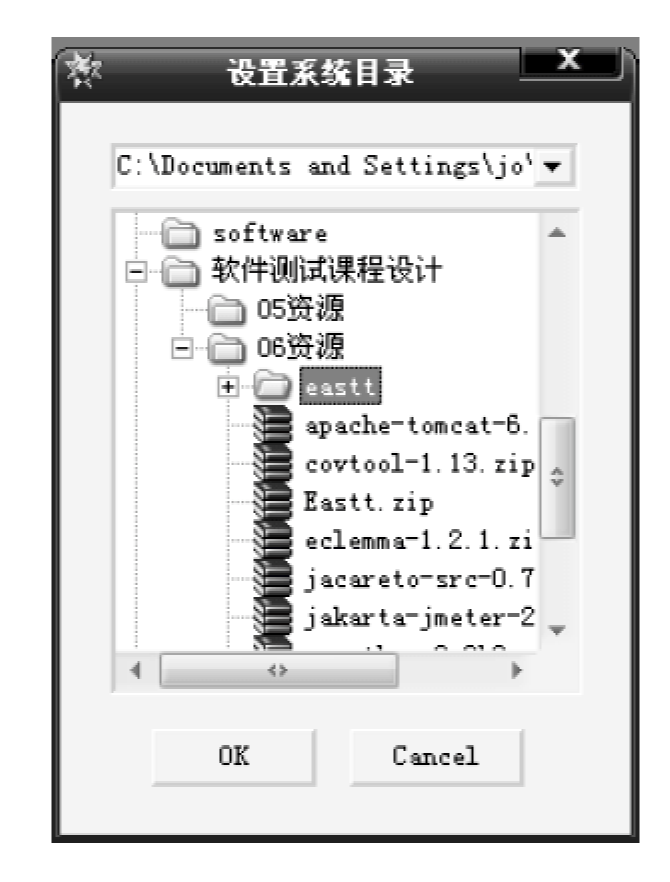
```
#include "ex02.h"
#include <iostream.h>
#include <stdlib.h>
void main()
{
int i;
cout<<"pls input make a choice\n";
cout<<"1 to do while iterating\n";
cout<<"2 to do for iterating\n";
cout<<"3 to do do-while iterating\n";
cout<<"4 to do exit iterating\n";
cin>>i;
 if (i==1)
{
      cout<<"entering while iteration\n";
```

```
        int z = 0;
        while (z<10)
        {
                z++;
                lbg l;
                int n;
                l.SetLbg(10);
        }
        system("pause");
}
else if(i= =2)
{
        cout<<"entering for iteration\n";
        for (i=0;i<10;i++)
        {
                cout<<"During for iteration\n";
                mbg o;
                int n;
                n = o.GetLbg();
        }
        system("pause");
}
else if(i= =3)
{
        int done=5;
        cout<<"entering do-while iteration\n";
        do{     int n;
                nbg p;
                n=p.GetLbg();
                cout<<"do-while";
                done--;
        }while(done);
        system("pause");
}
else
        system("pause");
lbg l;
int n;
return;
}

void lbg::SetLbg(int n)
{
libengu = n;
}
void mbg::SetLbg(int n)
```

```
{
libengu = n;
}
void nbg::SetLbg(int n)
{
libengu = n;
}
```

(1) 启动 EASTT，设置系统路径(EASTT 软件所在路径)，打开项目文件，如图 12-36 所示。



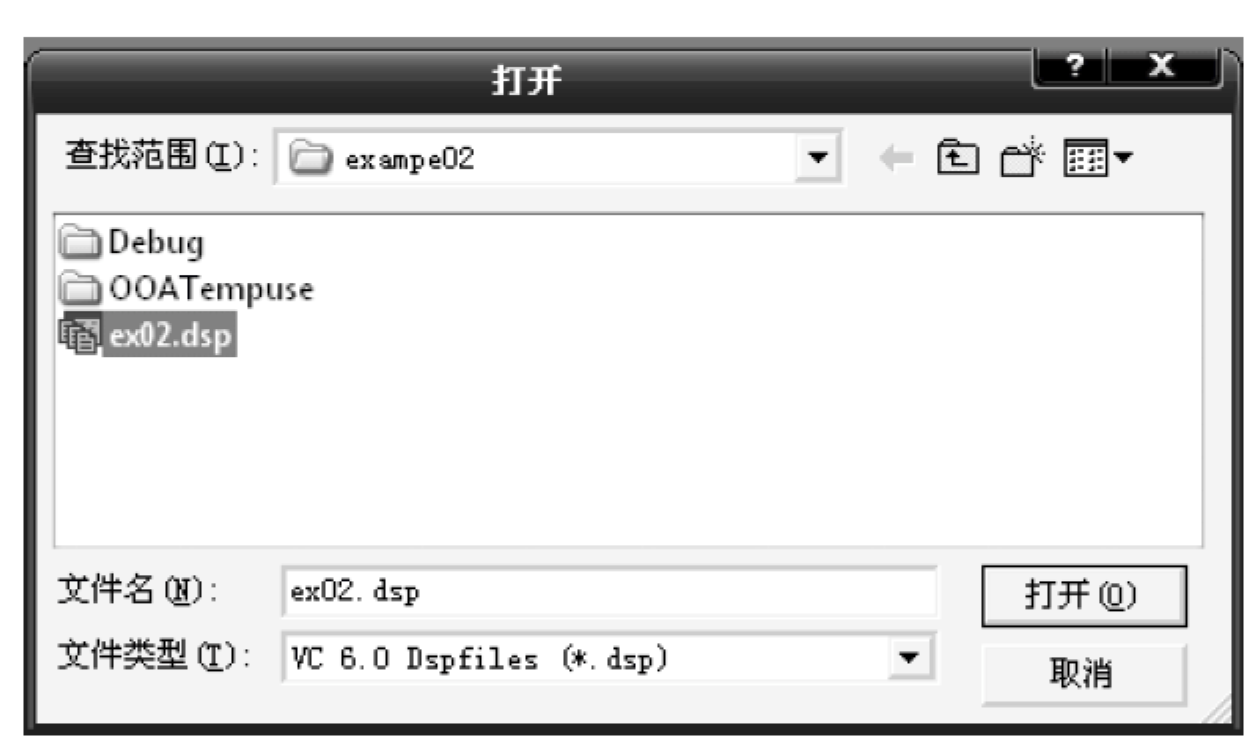


图 12-36　打开 ex02.cpp 的项目文件

注意：打开项目文件时，一定要使其后缀名为小写形式的 dsp，否则就会报错。

(2) 对 ex02.cpp 进行静态分析，如图 12-37 所示。

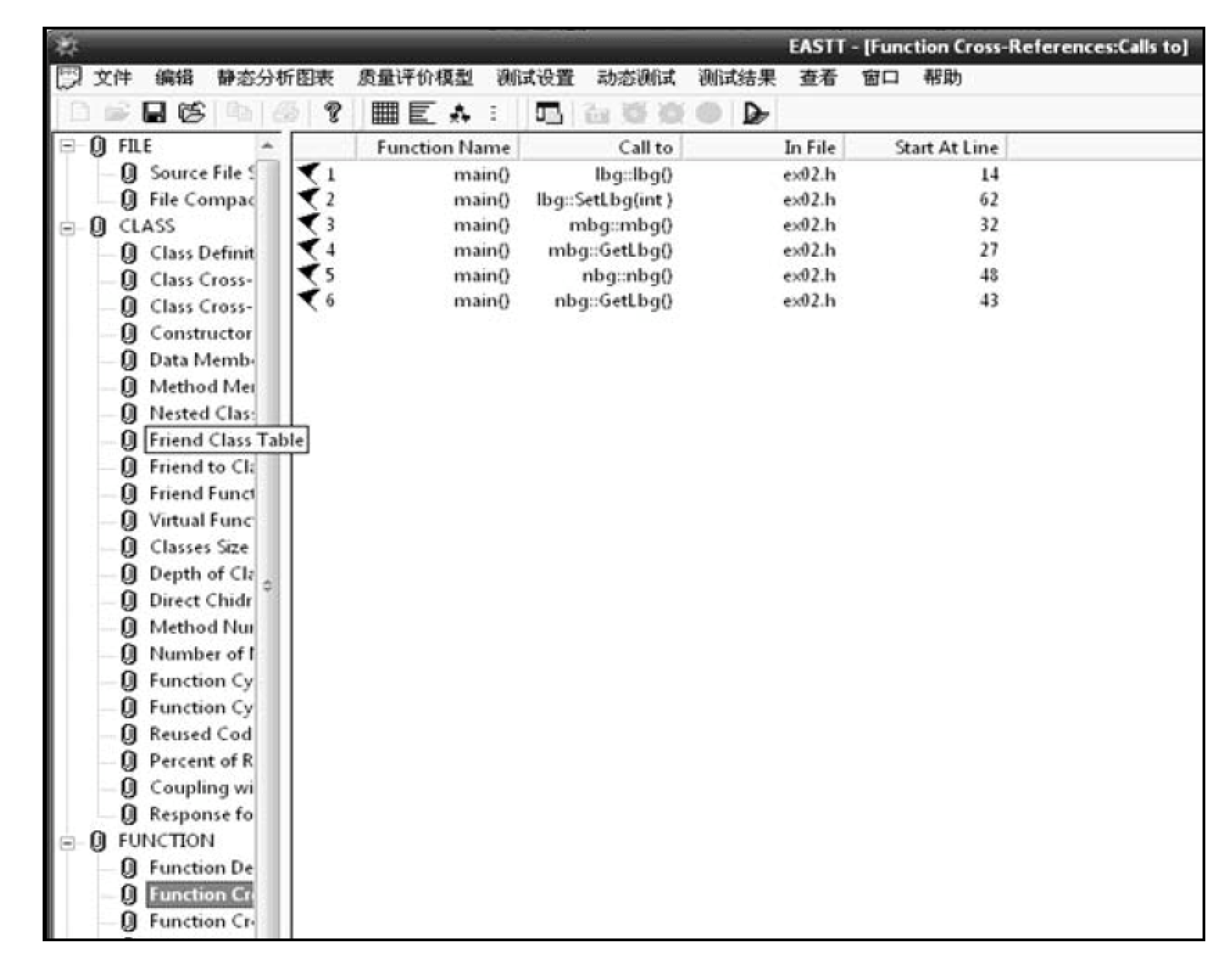


图 12-37　静态分析内容包含源文件、类、功能函数、变量及标号

(3) 对 ex02.cpp 进行结构浏览，如图 12-38 所示。列出每个类及其功能函数，以及被关注函数的代码。

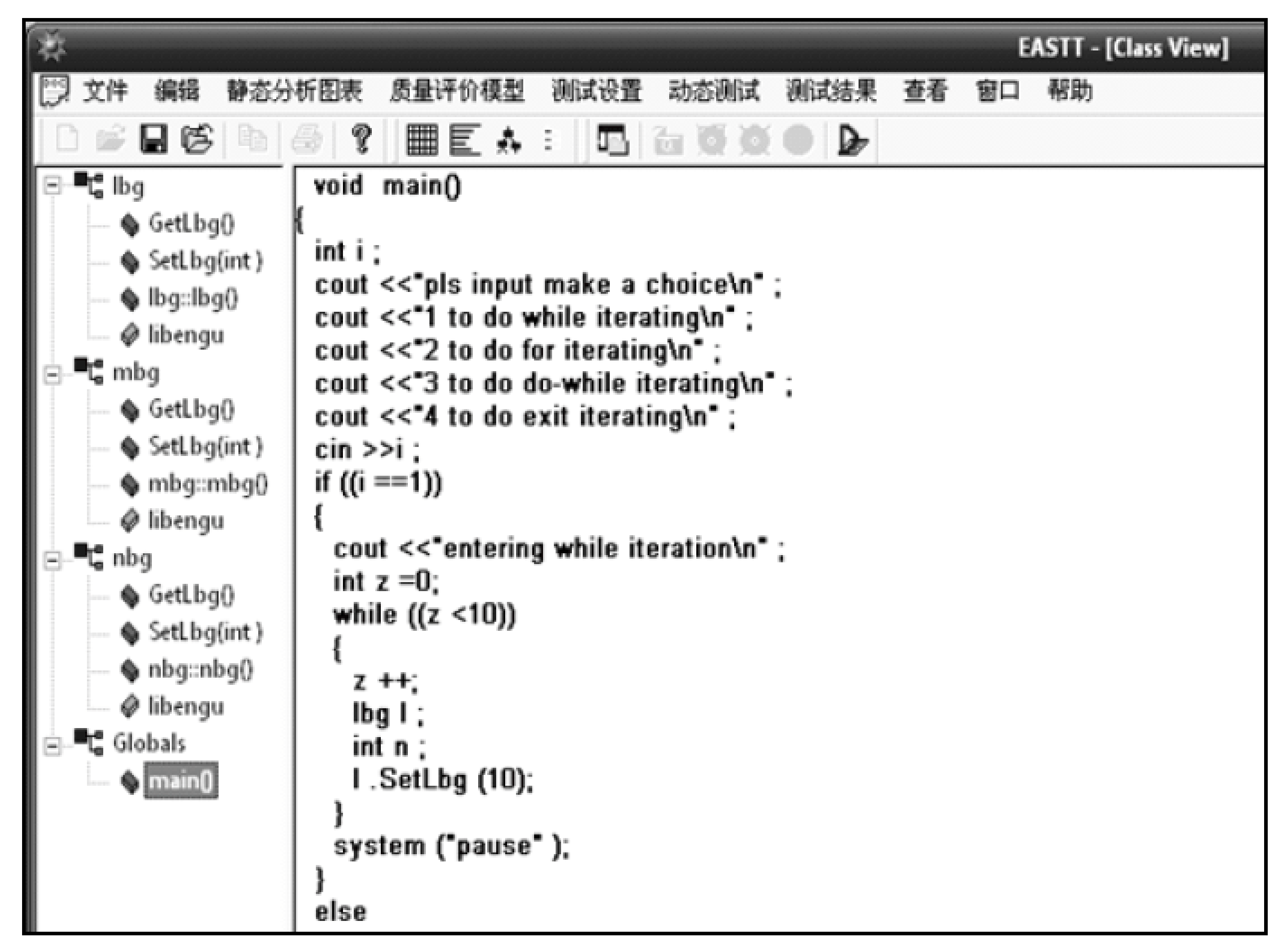


图 12-38　浏览 ex02.cpp 的结构

(4) 查看 ex02.cpp 的关系图，如图 12-39 所示。关系图分为类继承、函数调用、类/函数耦合以及 main 中函数调用情况。

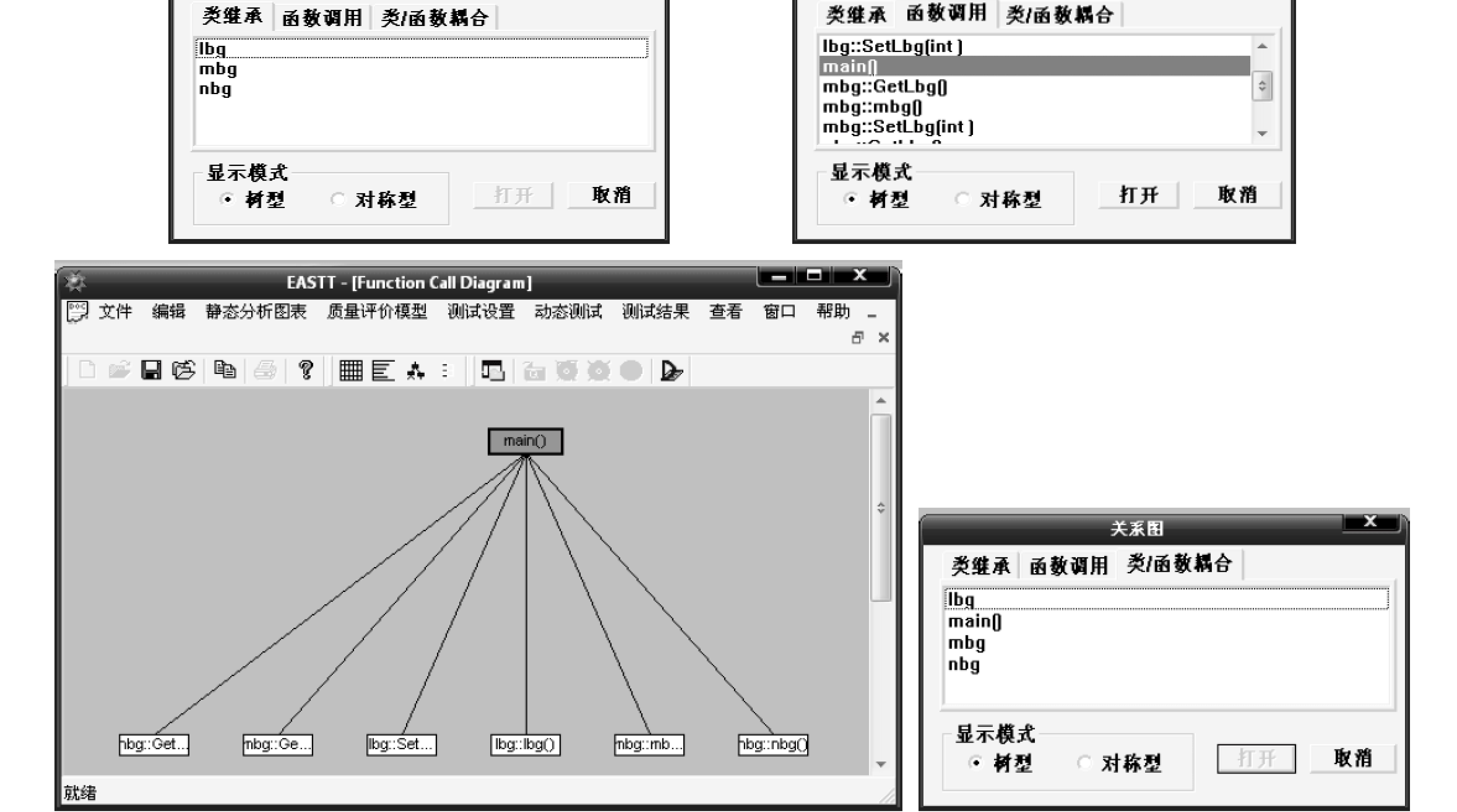


图 12-39　ex02.cpp 中的各种关系图

(5) 查看 ex02.cpp 的层次流程图，如图 12-40 所示。可以从类或文件中选择函数并查看其层次流程图，单击某条路径，该路径会变成红色，右边相对应执行的代码也将变为红色。





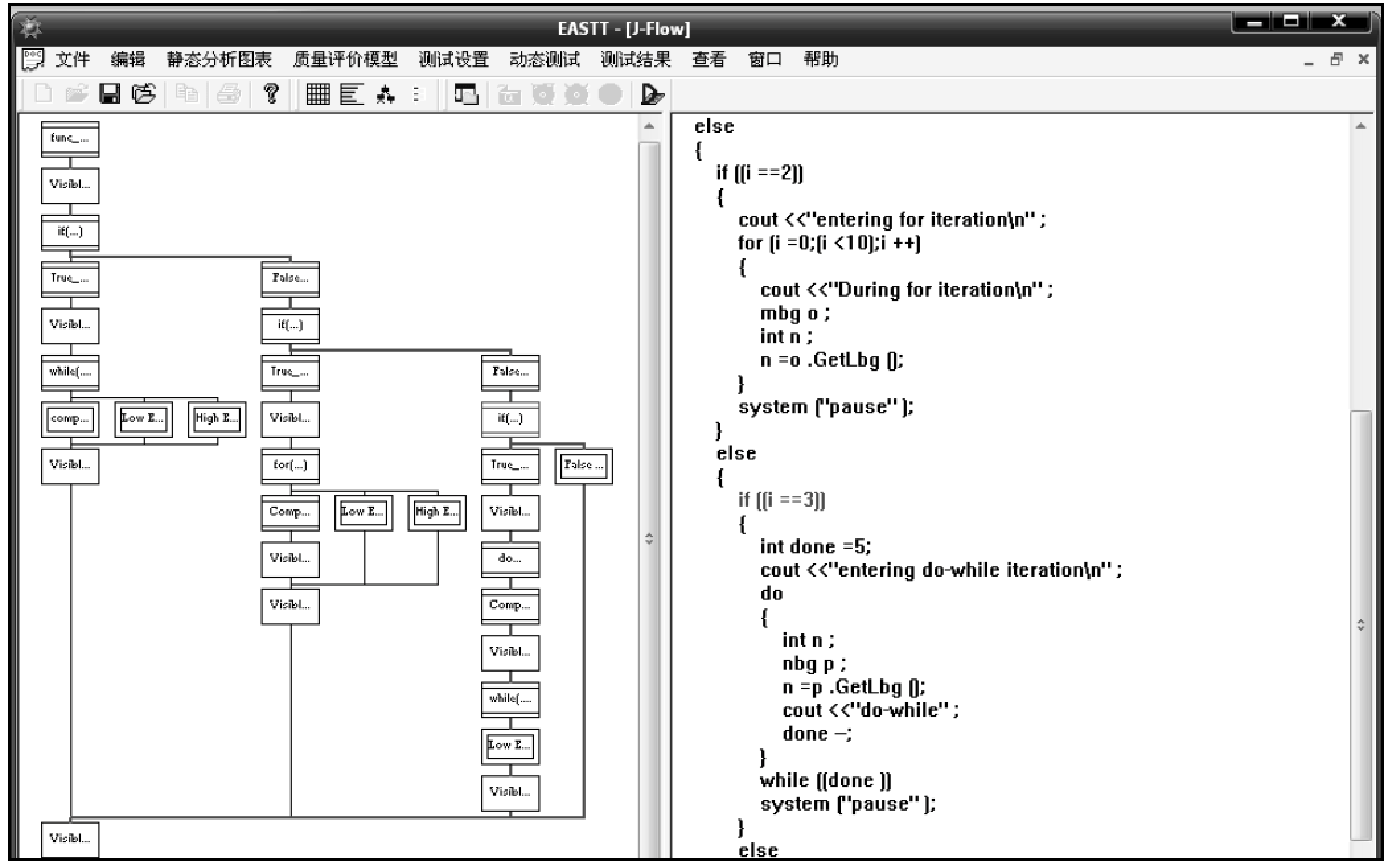


图 12-40　ex02.cpp 的层次流程图

(6) 载入三级质量评估度量文件(选择使用默认的 SQM 文件，也可以自定义 SQM 文件)，如图 12-41 所示。

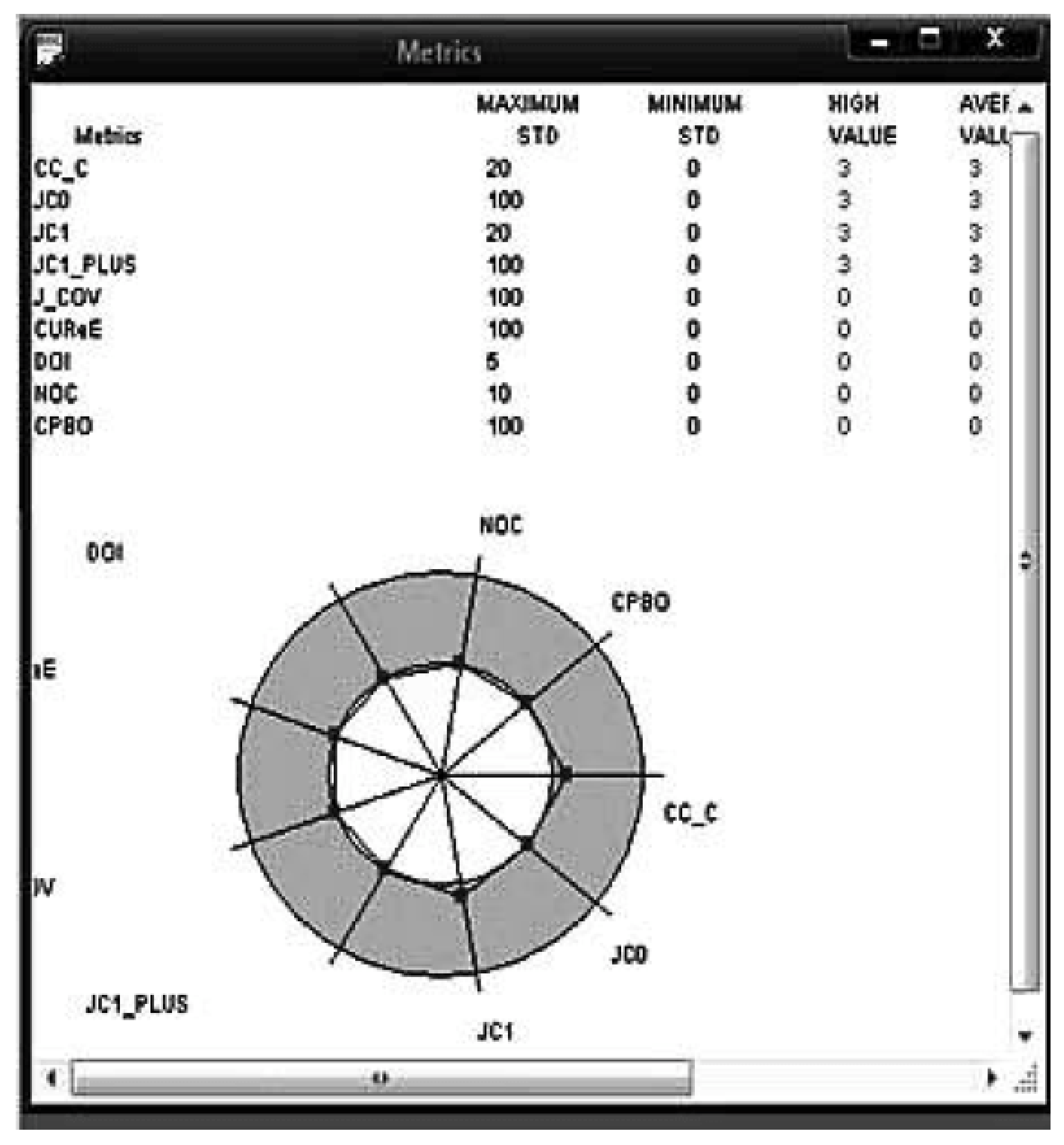


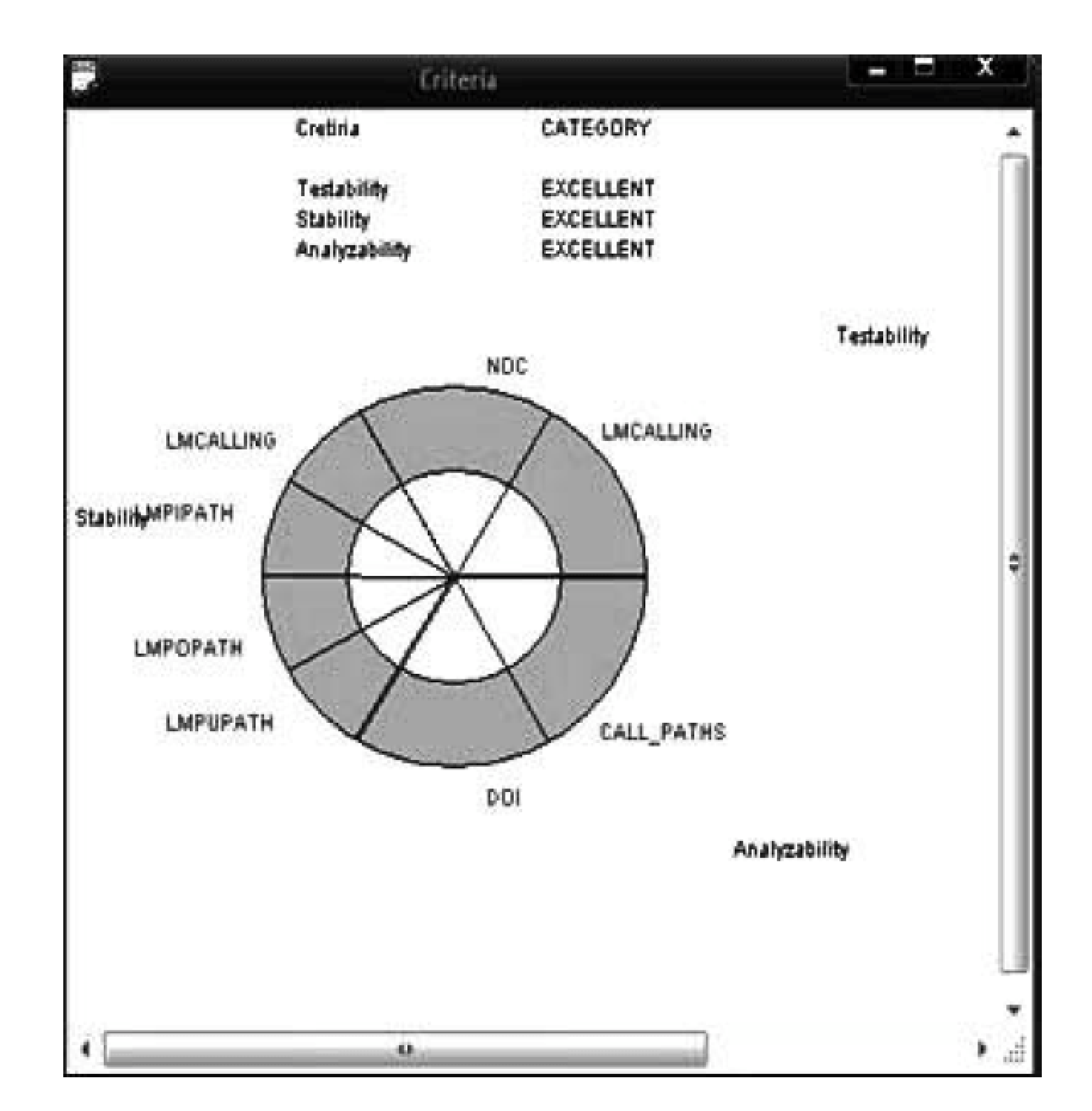




图 12-41　编辑三级质量评估度量文件

(7) 查看 ex02.cpp 的质量评估结果，如图 12-42 所示。





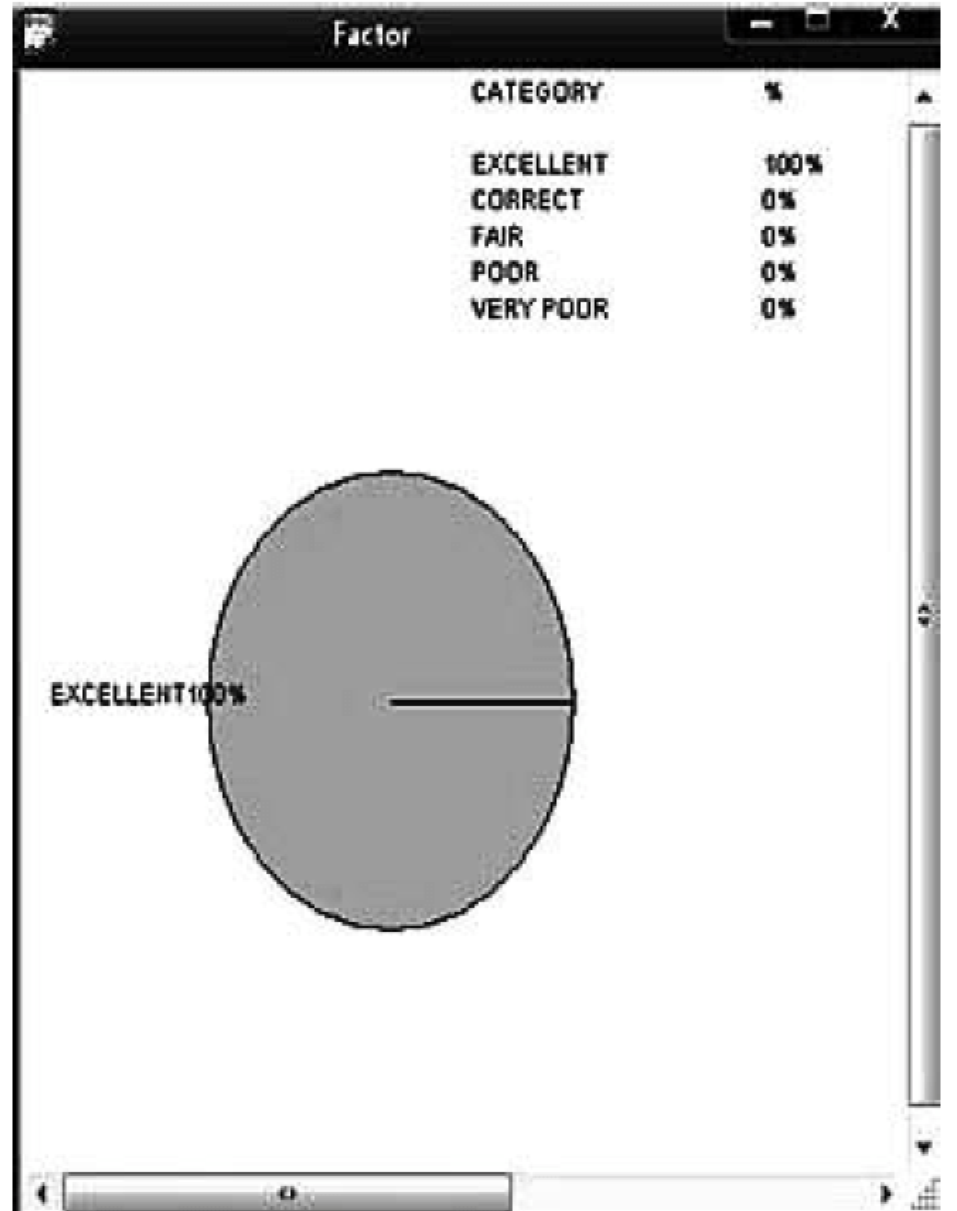


图 12-42　给出 ex02.cpp 的度量元、质量准则及质量因素

(8) 对 ex02.cpp 进行动态测试。

首先单击“动态初始化”，从控制台中选择要测试的程序，单击“动态插桩”。

插桩成功后，项目文件夹中会出现一个\OOAtempuse 文件夹，进入后再进入\target，有一个 VC++的工作空间 ex02.dsw。运行后，编译并生成可执行文件。

切换到 EASTT 软件平台，选择“编辑”|“添加新 TCase”命令。在 Command 文本框里输入待测的执行文件“ex02”，在 Work Direction 文本框中输入待测文件的路径，Parameter 是执行命令行时的参数，Description 则用来描述这次测试用例，如图 12-43 所示。



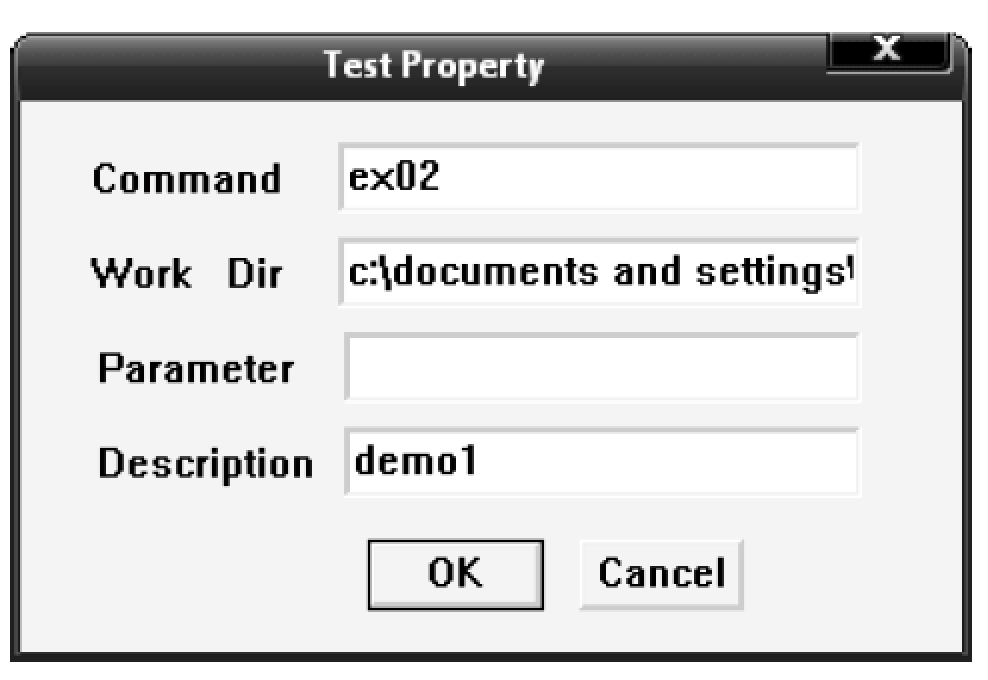
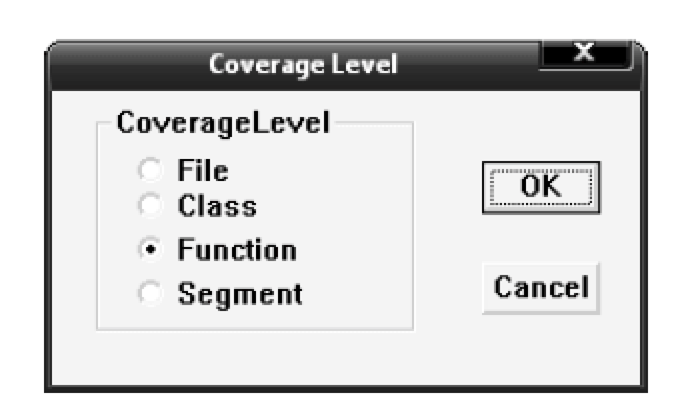
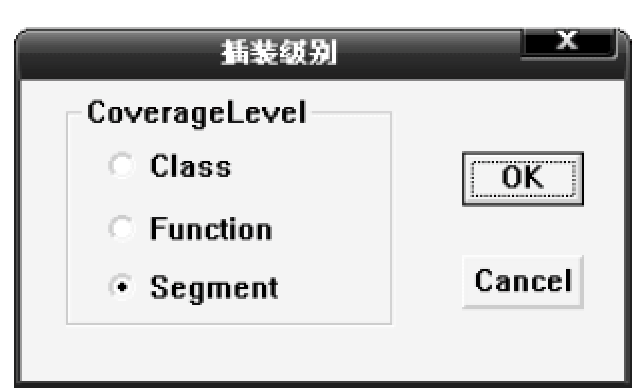


图 12-43　编写测试用例

(9) 设置 ex02.cpp 的覆盖测试级别并测试。

在创建了多个测试用例后，设置覆盖测试级别：插桩粒度设置成 Segment，覆盖级别设置为 Function。然后实施测试，分别输入 1、3、2、4 及非法值，分析最终测试结果，如图 12-44 所示。





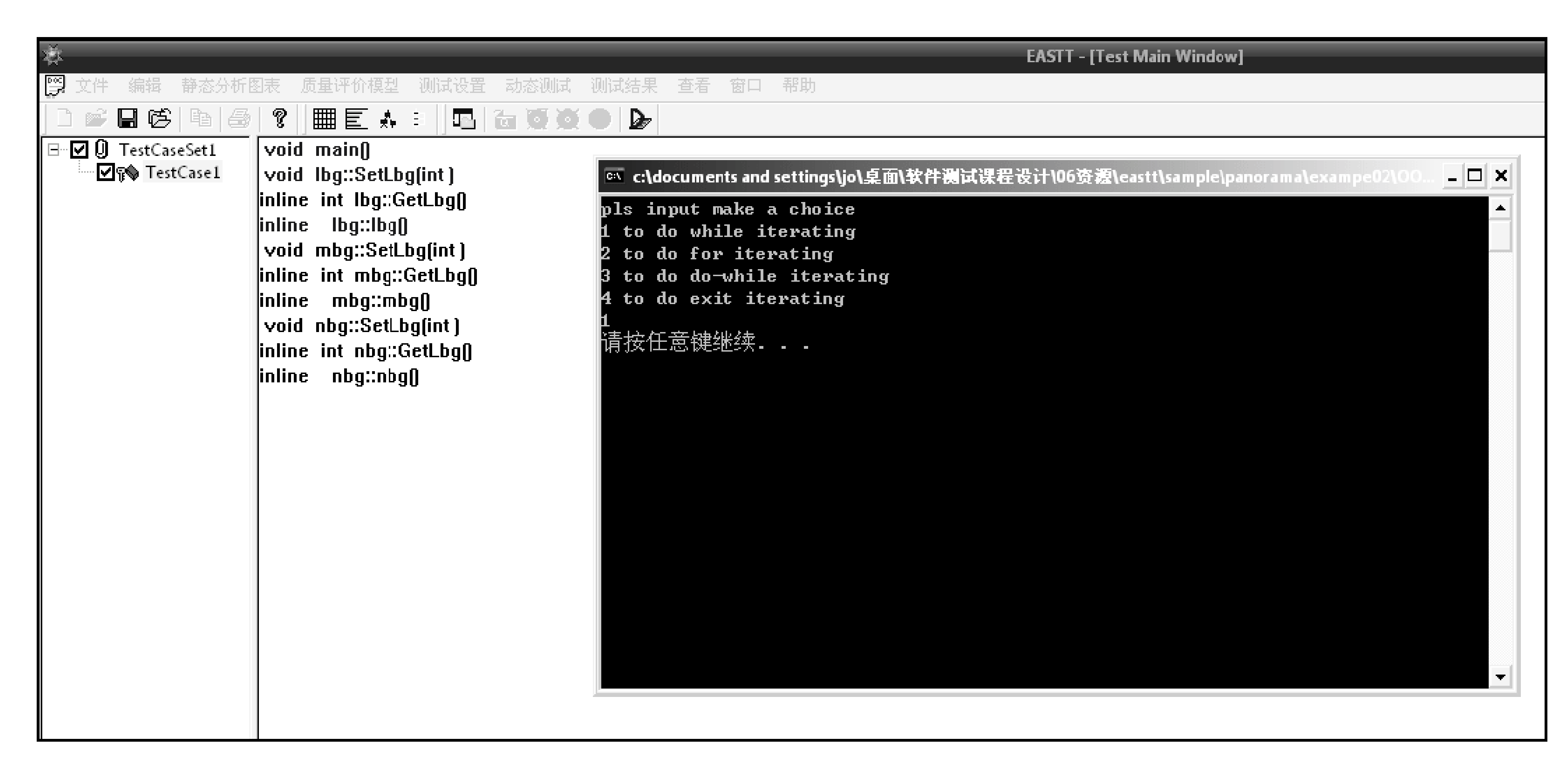
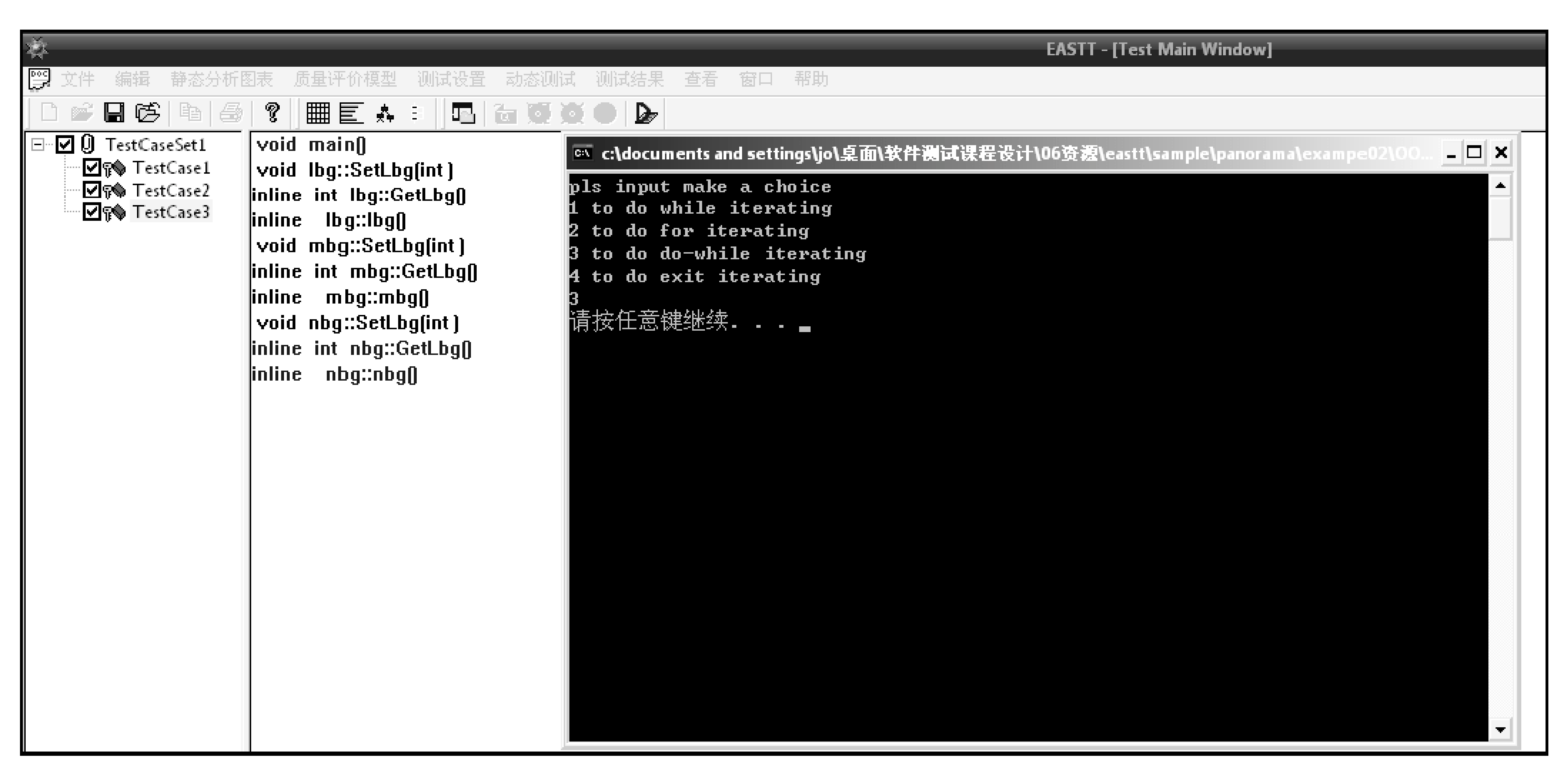
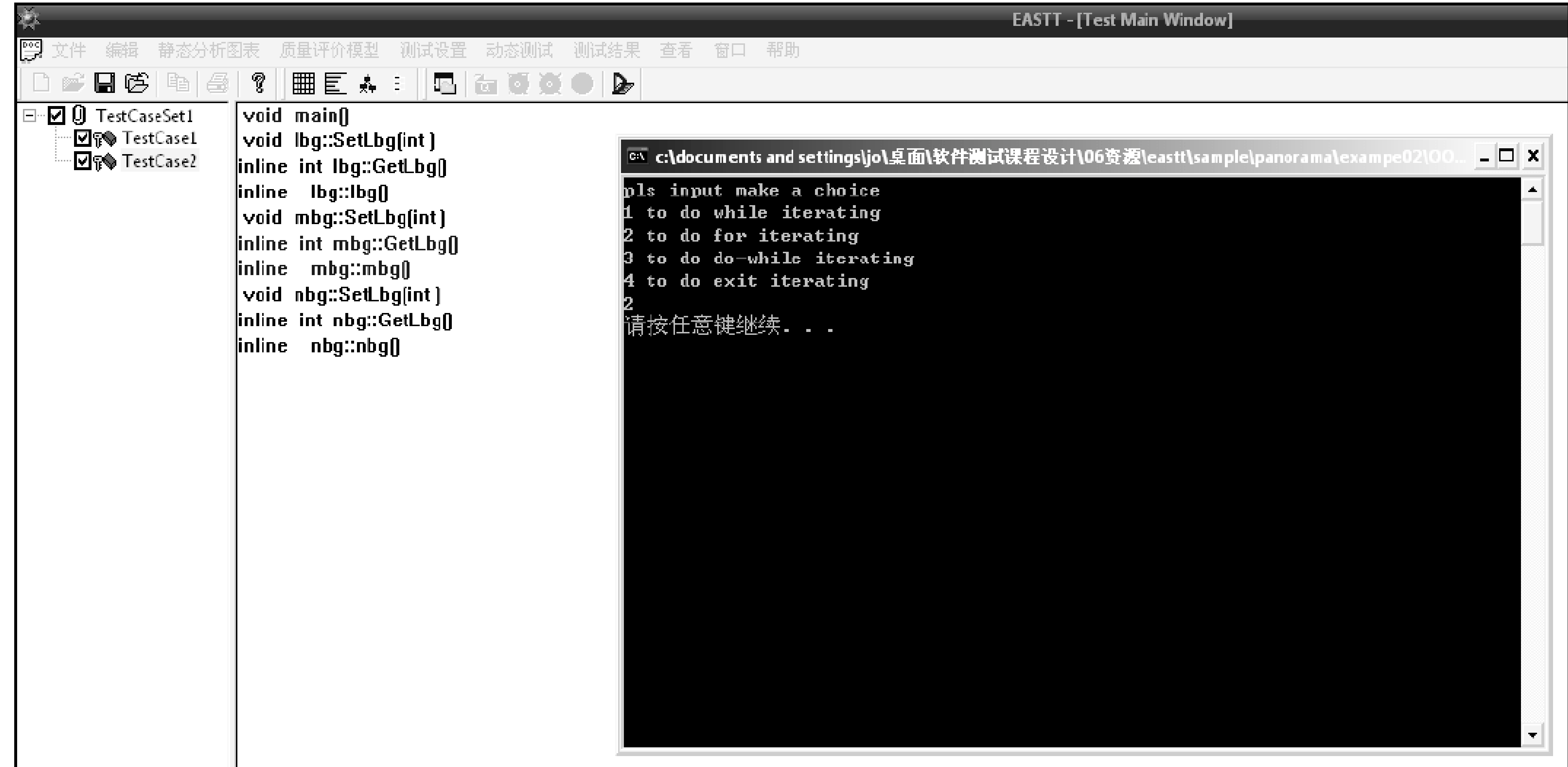


图 12-44　测试实施过程 1(插桩粒度设置成 Segment，覆盖级别设置为 Function)


EASTT - [Test Main Window]
文件 编辑 静态分析图表 质量评价模型 测试设置 动态测试 测试结果 查看 窗口 帮助
TestCaseSet1
TestCase1
TestCase2
TestCase3
void main()
void lbg::SetLbg(int)
inline int lbg::GetLbg()
inline lbg::lbg()
void mbg::SetLbg(int)
inline int mbg::GetLbg()
inline mbg::mbg()
void nbg::SetLbg(int)
inline int nbg::GetLbg()
inline nbg::nbg()
pls input make a choice
1 to do while iterating
2 to do for iterating
3 to do do-while iterating
4 to do exit iterating
3
请按任意键继续. . .


EASTT - [Test Main Window]
文件 编辑 静态分析图表 质量评价模型 测试设置 动态测试 测试结果 查看 窗口 帮助
TestCaseSet1
TestCase1
TestCase2
void main()
void lbg::SetLbg(int)
inline int lbg::GetLbg()
inline lbg::lbg()
void mbg::SetLbg(int)
inline int mbg::GetLbg()
inline mbg::mbg()
void nbg::SetLbg(int)
inline int nbg::GetLbg()
inline nbg::nbg()
pls input make a choice
1 to do while iterating
2 to do for iterating
3 to do do-while iterating
4 to do exit iterating
2
请按任意键继续. . .

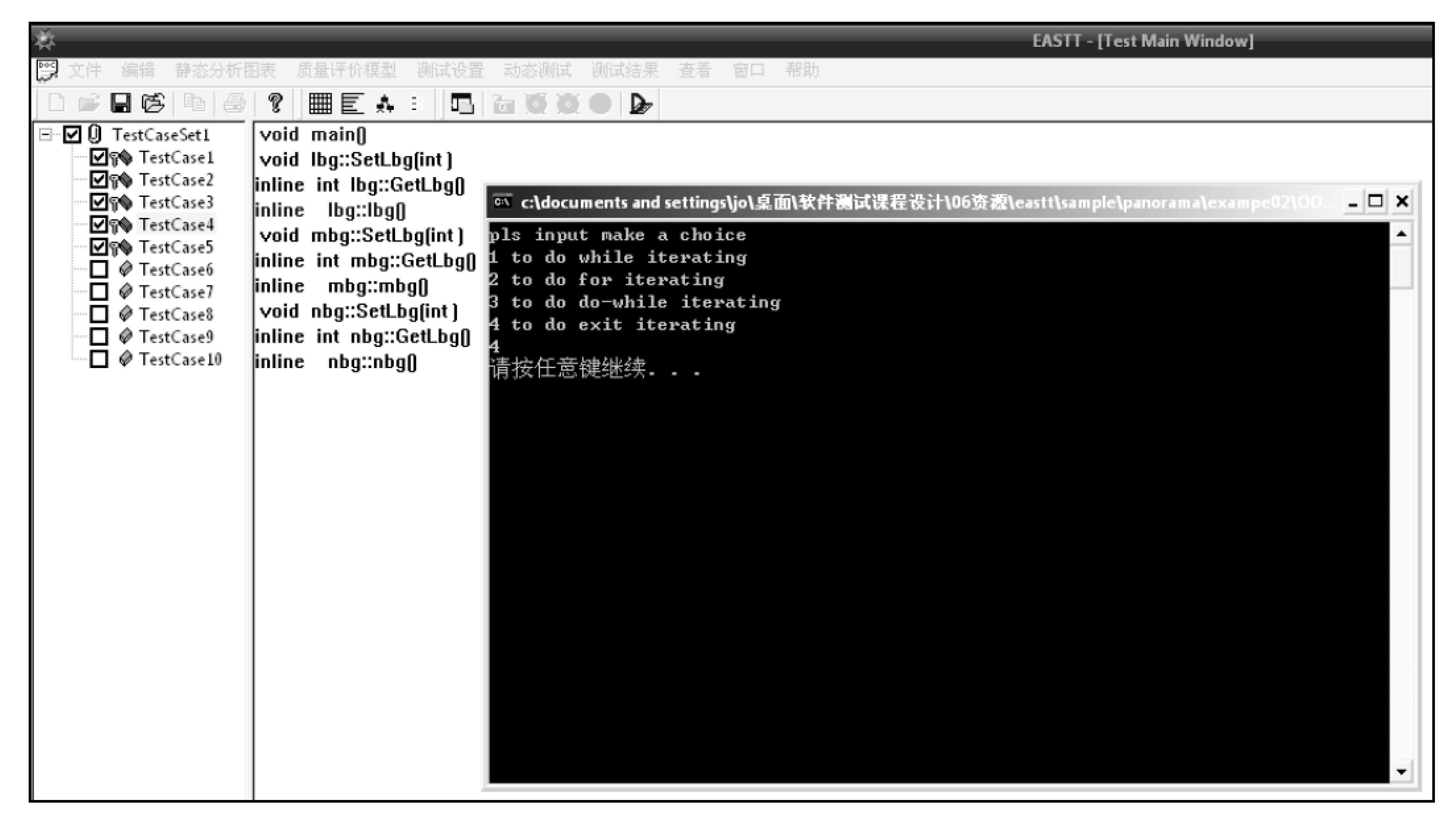

EASTT - [Test Main Window]
文件 编辑 静态分析图表 质量评价模型 测试设置 动态测试 测试结果 查看 窗口 帮助
TestCaseSet1
TestCase1
TestCase2
TestCase3
TestCase4
TestCase5
TestCase6
TestCase7
TestCase8
TestCase9
TestCase10
void main()
void lbg::SetLbg(int)
inline int lbg::GetLbg()
inline lbg::lbg()
void mbg::SetLbg(int)
inline int mbg::GetLbg()
inline mbg::mbg()
void nbg::SetLbg(int)
inline int nbg::GetLbg()
inline nbg::nbg()
pls input make a choice
1 to do while iterating
2 to do for iterating
3 to do do-while iterating
4 to do exit iterating
4
请按任意键继续. . .

图 12-44　(续)

如果覆盖分析级别选择为 Class，结果将会如图 12-45 所示。



图 12-45　测试实施过程 2(插桩粒度设置成 Segment，覆盖级别设置为 Class)

同理，如果覆盖分析级别选择为 File，结果将会如图 12-46 所示。



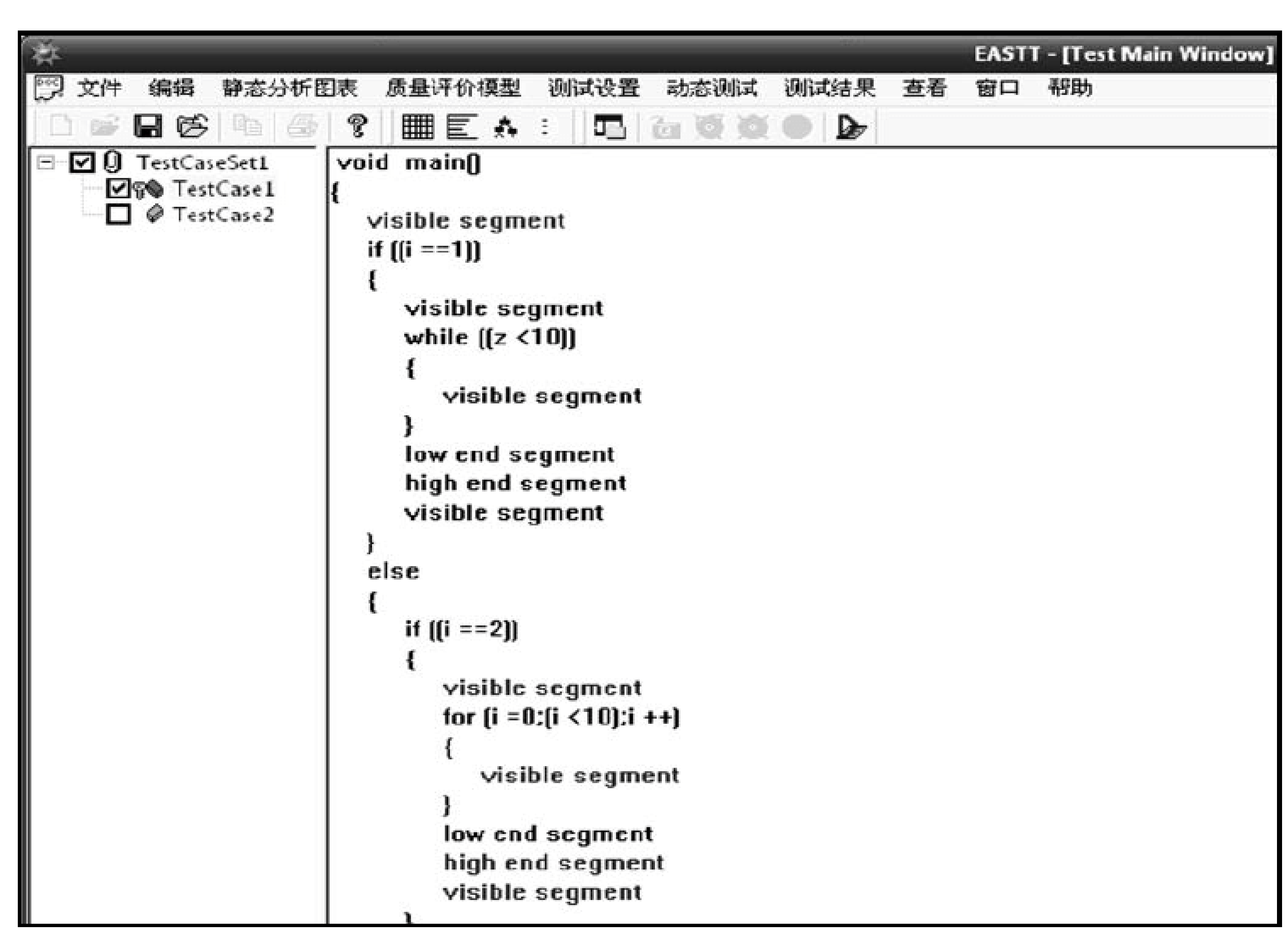


图 12-46　测试实施过程 3(插桩粒度设置成 Segment，覆盖级别设置为 File)

分析测试结果时，要注意的是：①左边图标为粉红色的表示测试用例已执行过，其他颜色的则表明没有执行过；②右边的蓝色部分是已覆盖部分，其他颜色的则是没有覆盖过的。

(10) 查看 ex02.cpp 的动态表格分析。

根据动态测试平台中复选框中打“√”的测试用例集合进行统计，在表格中进行显示，单击后面红色的序号，表格就会弹出，如图 12-47 所示。

(11) 查看 ex02.cpp 测试用例最小化。

这里做了 10 个测试用例，使用测试用例最小化功能可以去除重复的测试用例，并得出用例最小化，如图 12-48 所示。

(12) 动态测试效率分析。

显示测试用例是否被测试，以及其语句段的覆盖效率，如图 12-49 所示。

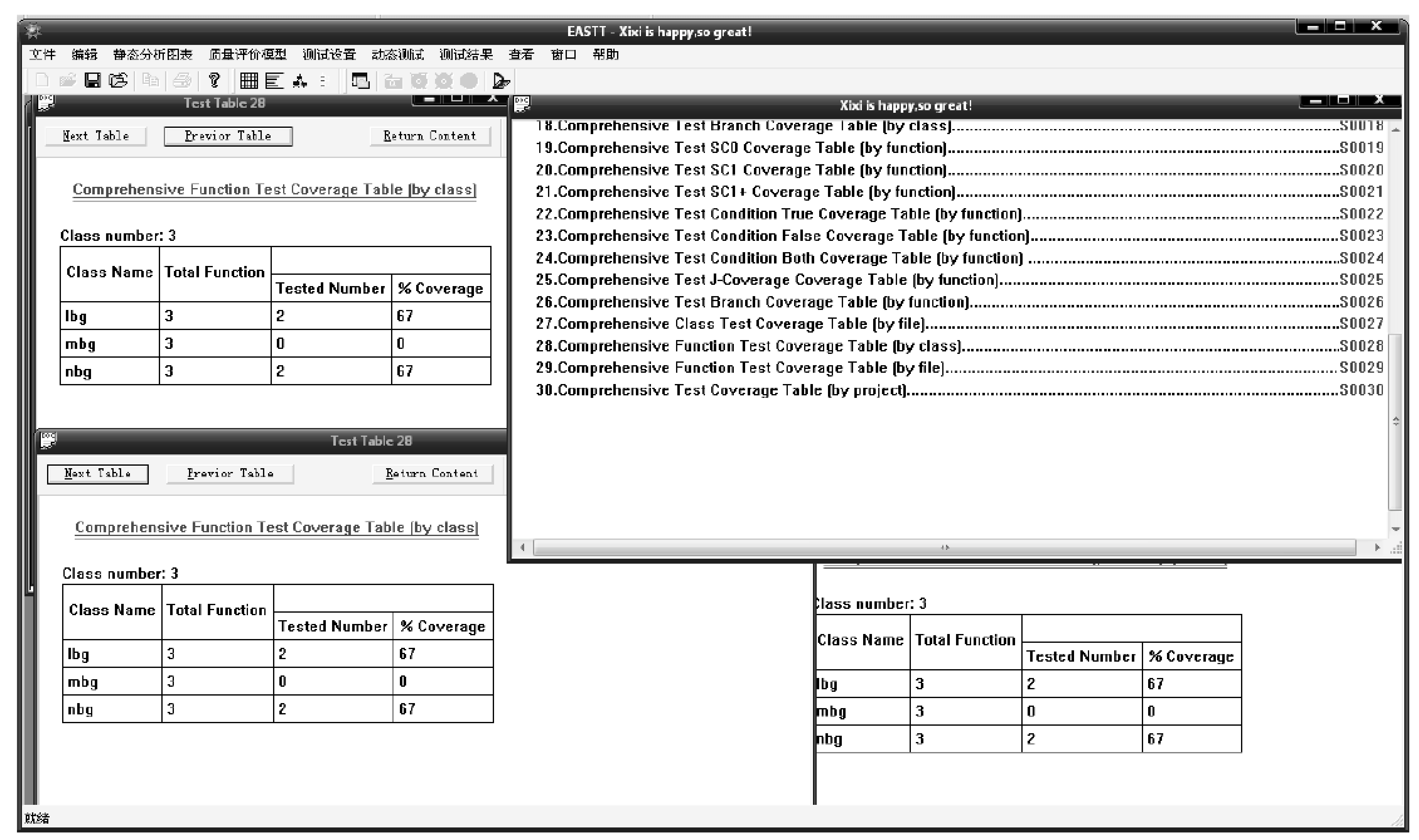


图 12-47　ex02.cpp 测试结果表格分析

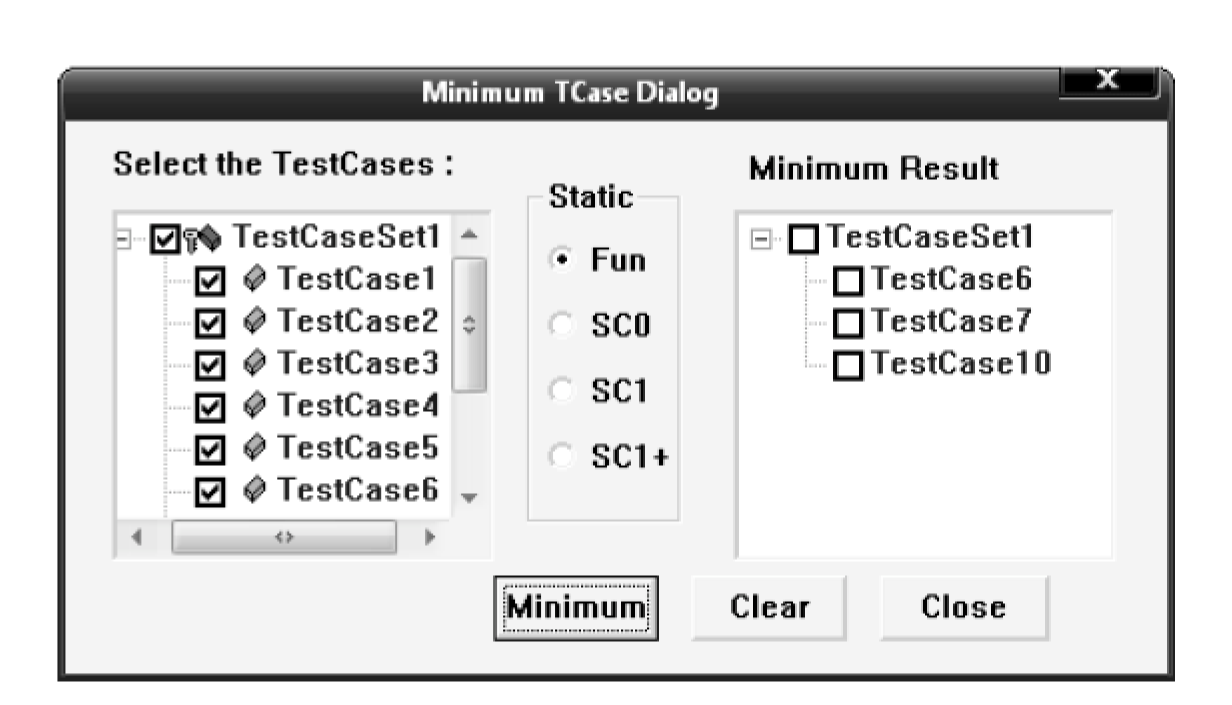
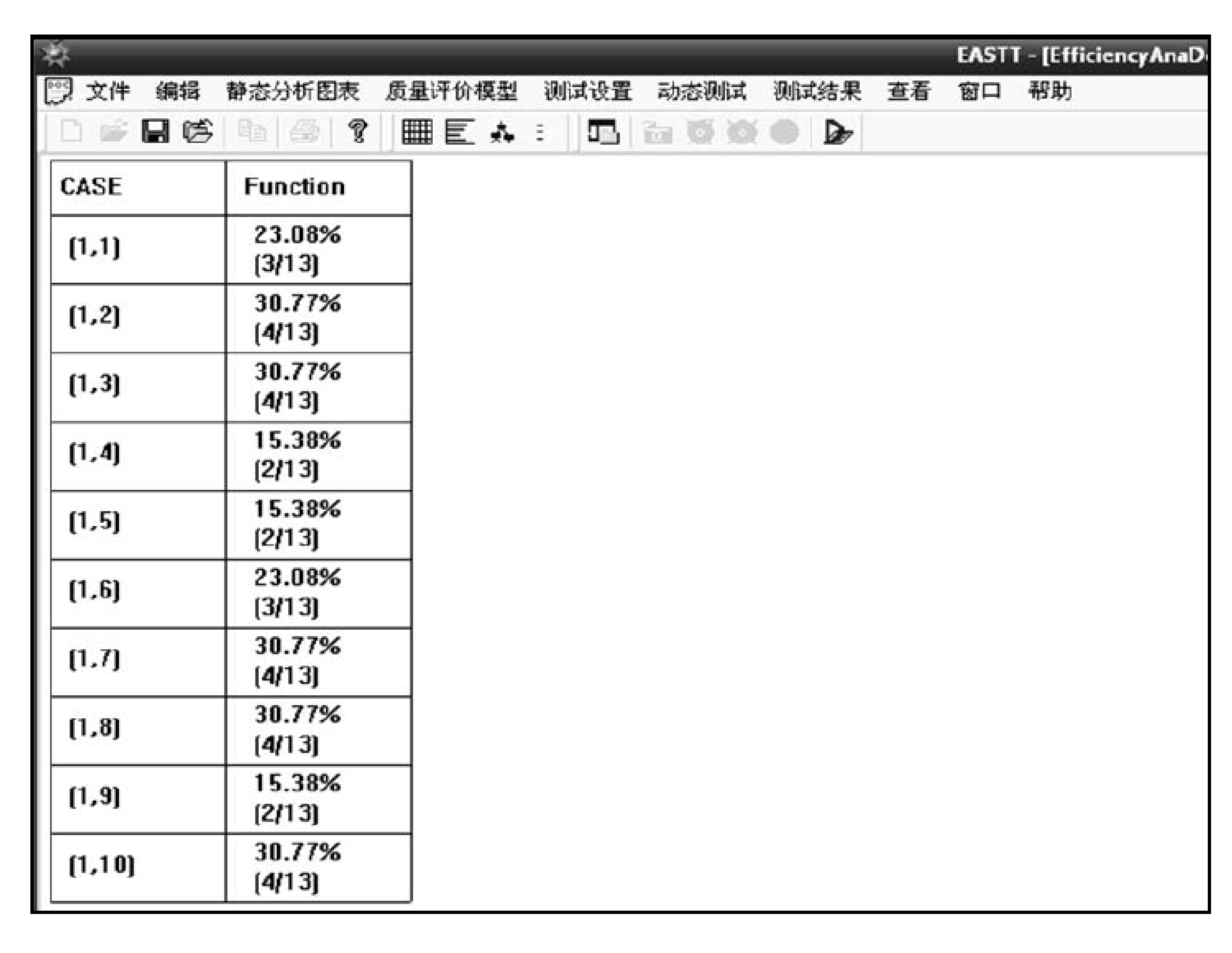


图 12-48　ex02.cpp 测试用例最小化



CASE	Function
[1,1]	23.08% (3/13)
[1,2]	30.77% (4/13)
[1,3]	30.77% (4/13)
[1,4]	15.38% (2/13)
[1,5]	15.38% (2/13)
[1,6]	23.08% (3/13)
[1,7]	30.77% (4/13)
[1,8]	30.77% (4/13)
[1,9]	15.38% (2/13)
[1,10]	30.77% (4/13)

图 12-49　测试用例效率统计

(13) 查看动态 PPP。

选择 main 函数，图 12-50 显示了 main 函数与其调用的函数，中间的数字代表调用的次数。



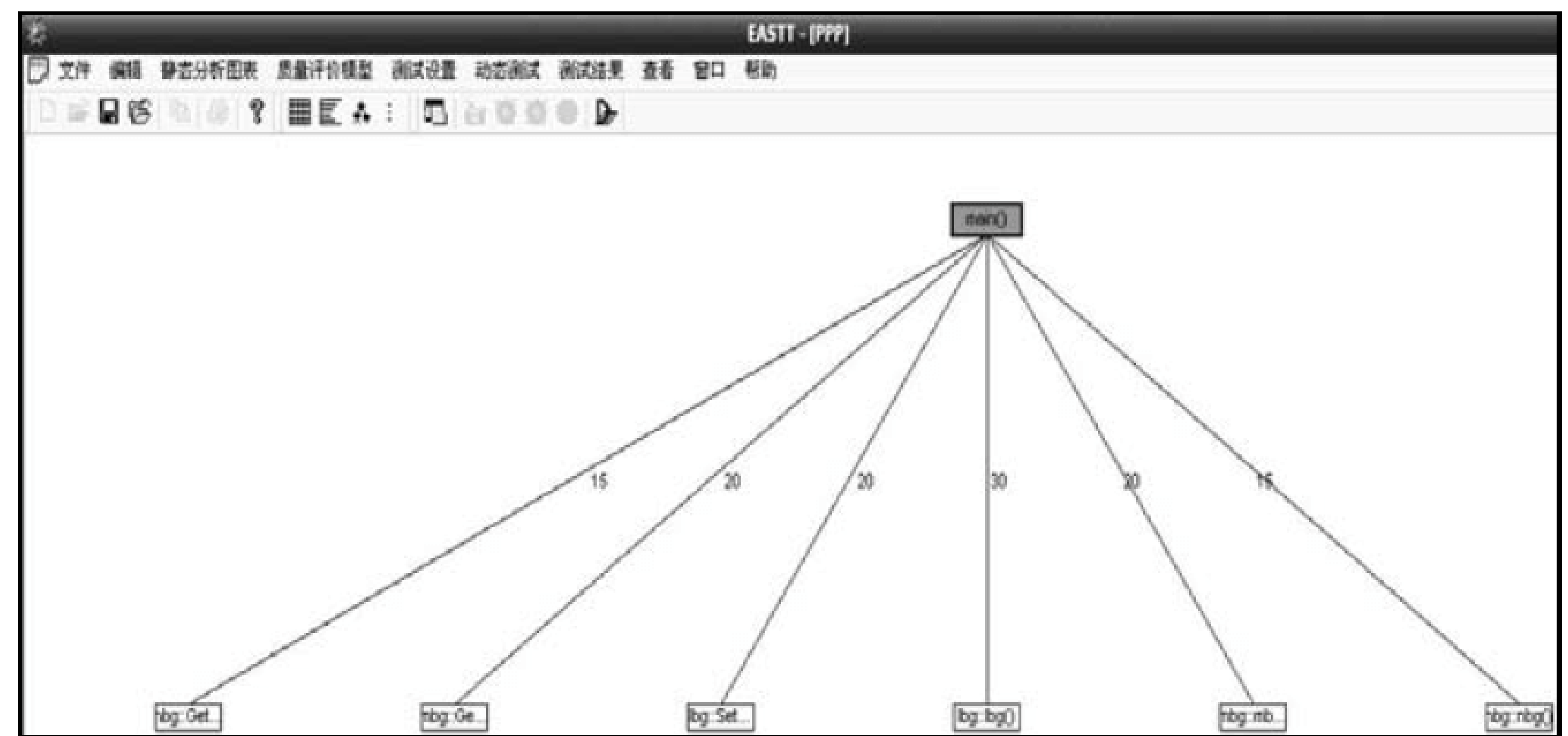


图 12-50　ex02.cpp 执行的动态 PPP

(14) 查看 ex02.cpp 的动态流程图。

从类或文件里选择 main()函数，显示其动态流程分析图，如图 12-51 所示。

(15) 检查测试开销。

检查结果以插桩代码的运行时间占被测程序总运行时间的百分比的形式显示，如图 12-52 所示。

(16) 生成测试报告。

生成 HTML 格式的文档，其内容包括被测文件列表、静态分析结果表格、动态测试覆盖表格等，如图 12-53 所示(可以在项目文件夹中找到)。





图 12-51　ex02.cpp 中 main 函数的动态流程图

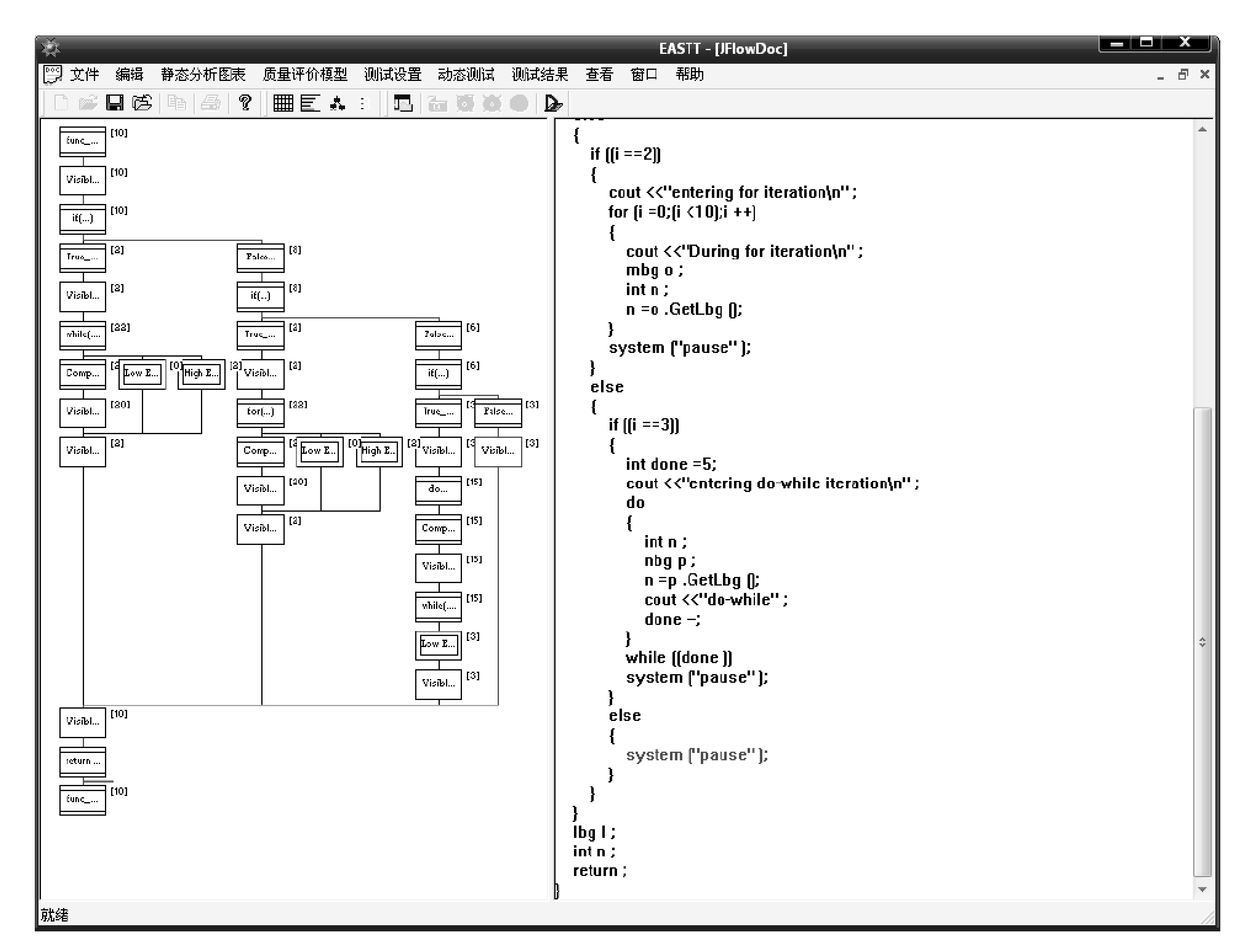


图 12-51　(续)



图 12-52　ex02.cpp 中插桩代码后造成的测试开销

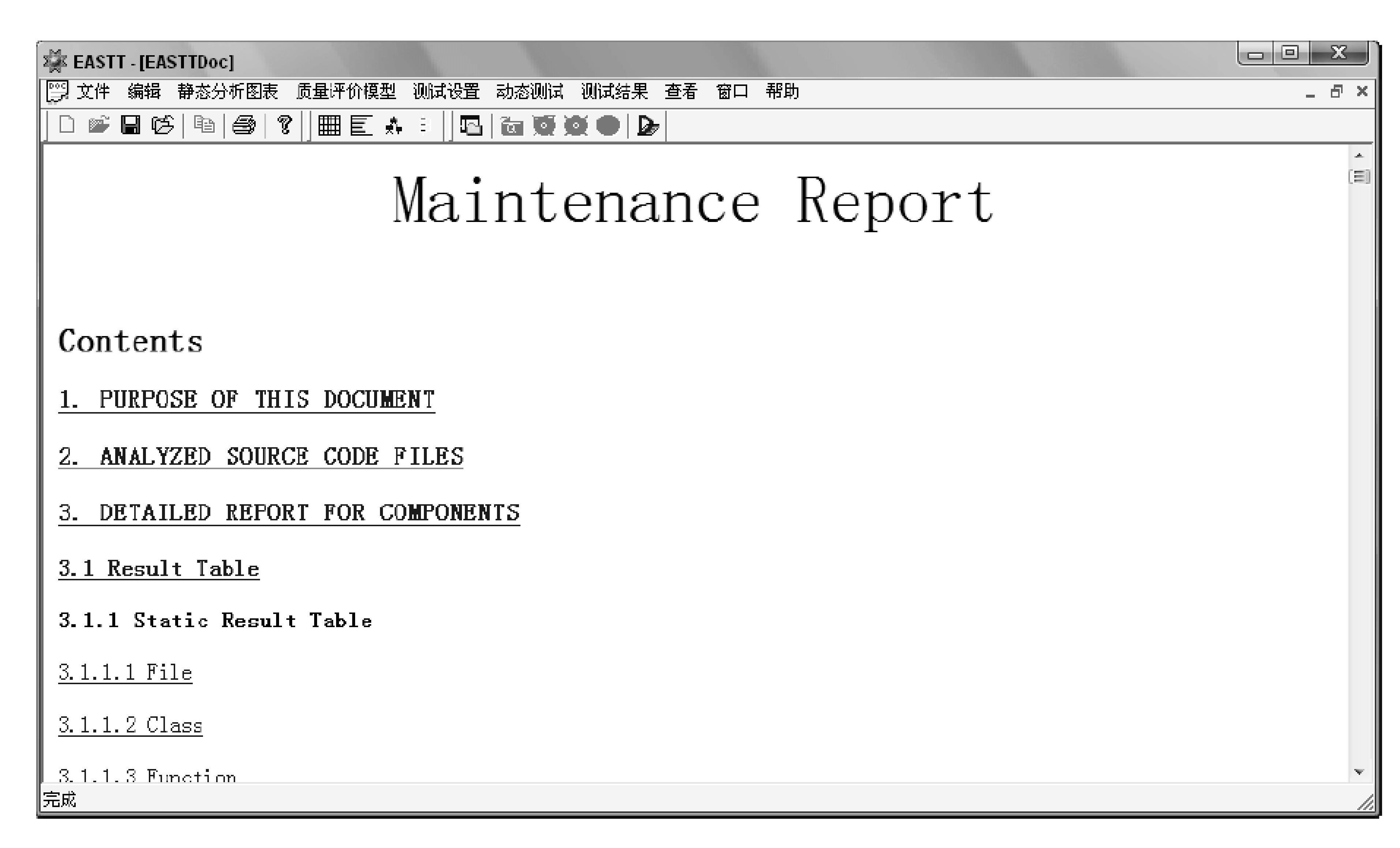


图 12-53　ex02.cpp 测试完成后生成的测试报告

12.5　EASTT 应用小结

EASTT 作为一种软件评测工具，可谓功能强大、测试全面。但是它依然存在着很多问题，首先它仍然停留在早期的 Windows 9x、Windows NT 4.0、Windows 2000 平台环境，仅初步支持 VC 6 的应用测试。而且 EASTT 所宣扬的对实时嵌入式应用软件测试的支持是建立在早已消失的 PSOS 实时操作系统上。因此 EASTT 目前不具备可用性。其次是 EASTT 系统的稳定性很不好，错误很多。对于有些被测项目，系统往往不能够成功加载，或者加载完毕后执行某些功能模块时系统出现崩溃。另外，EASTT 软件的部分功能实现有较为严重的错误或不合理之处。如表格分析中的覆盖率计算，每关闭打开一次就会将覆盖率与上一次分析的结果累加，导致覆盖率经常会超过 100%，从而失去了覆盖率的意义。测试用例最小化功能则根本无法正常工作，无论选择什么样的测试用例、什么样的最小化级别，所得到的结果都只有最后一个测试用例。还有，文档生成功能会直接导致程序崩溃退出等。最后，EASTT 在进行质量分析时，很多评价指标不能使用，或者计算结果是错误的。

总之，EASTT 作为一个测试工具，其自身没有很好地被测试，这对倡导软件工程和软件质量的先驱者多少是一个讽刺；作为 863 科研项目公布这样的源码也是很不严谨的，这重挫了国人对国内开源软件的信心；最后，作为 Logiscope 和 Panorama++的模仿秀，EASTT 也是不成功的，感觉画虎不成反类犬。

当然，EASTT 也有它的教育意义：①软件评测知识的学习和应用，毕竟 EASTT 提供了一个可以进行评判的软件评测平台；②思想的转变，我们不需要标榜有多先进的科研成果，我们需要的是真正可用的软件产品或软件工具；③对软件产品的真正理解，软件的配套文档实质上是软件不可缺少的重要组成部分；④软件维护和升级的重要性，软件产品不进行维护、不及时升级，这个软件产品迟早会被人抛弃；⑤作为反面教材，EASTT 提供了一个很好的被测软件；⑥国内有志之士可在 EASTT 基础上做进一步工作，使之走得更远，推出真正的国产软件产品，让教师和学生在使用这些软件产品进行实践教学的时候，不再抱有遗憾。

实 验 习 题

1. 应用 EASTT 工具对售货机程序进行全面的综合评测，并详细描述测试过程和测试结果。

2. 以 EASTT 作为被测程序进行全面的手工测试，并将测试中发现的问题用 Mantis 和 TestLink 进行缺陷管理和测试管理。

参 考 文 献

[1] 杨静波，田悦峰．软件测试过程管理实践. 51testing，2007-5-17.

[2] 朱三元．软件质量及其评价技术．北京：清华大学出版社，1990.

[3] 孙志安．软件复杂性的度量与控制．计算机世界报，1997, (42).

[4] 陈世基．基于 McCabe 的软件复杂性度量与控制策略．电脑知识与技术，2007, 1(05).

[5] 谷照燕．一种新的软件测试方法．赤峰学院学报，2006, 22(1).

[6] 郑人杰．计算机软件测试技术．北京：清华大学出版社，1992.

[7] 刘海，郝克刚．软件缺陷数据的分析方法及其实现．计算机科学，2008, 35(8).

[8] 殷广丽．浅析软件测试管理及缺陷管理．山东教育学院学报，2005, (5).

[9] 王用杰等．软件测试管理的设计与实践．四川大学学报(自然科学版)，2004, 41(增刊).

[10] 高慧英．软件测试管理及其工程应用．计算机与数字工程，2007, 35(1).

[11] 尹相乐等．软件缺陷分类的研究．计算机工程与设计，2008，29(19).

[12] 路莹，马立权．软件缺陷跟踪管理系统设计．大连轻工业学院学报，2005, 24(3).

[13] 软件测试网中的 April 测试家园. TestManager 与 TestDirector 的优缺点.
http://www.51testing.com/?uid-85168-action-viewspace-itemid-10618，2007-5.

[14] 王德胜，宫云战．论软件缺陷．计算机应用研究，2008, 25(12).

[15] 李丹．软件缺陷报告．电子质量，2003, (07).

[16] 领测软件测试网．如何使用开源缺陷管理工具 Mantis 跟踪 Bug.
http://www.ltesting.net/html/77/n-161477.html，2009-3.

[17] 蔡琰．使用开源缺陷管理工具 Mantis 实施缺陷跟踪．测试时代采编,
http://www.ltesting.net/html/72/n-161872.html，2009-4-17.

[18] 51Testing 软件测试论坛. Mantis 缺陷管理系统使用说明.
http://bbs.51testing.com/thread-8352-1-1.html，2005-2-2.

[19] 小龙虾的博客．缺陷管理工具比较.
http://www.cnitblog.com/Candy/archive/2007/10/13/34814.aspx，2007-11-28.

[20] ITPUB 个人空间网中的 CloudSpace.Mantis 安装与配置(Windows+MySQL+PHP+IIS).
http://blog.itpub.net/14789789/viewspace-436099, 2008-8.

[21] 软件测试时代网．缺陷管理中的 Mantis 的应用.
http://www.testage.net/html/48/n-648.html，2009-4-3.

[22] 中国 Mantis 社区. Mantis 资源中心. http://www.mantis.org.cn/index.php.

[23] 软件测试网．测试管理流程简介. http://www.51testing.com/html/40/n-67240.html,2007-11-21.

[24] 徐异婕．使用 TestLink 进行测试管理. http://www.testage.net/html/79/n-779.html，2009-5-26.

[25] 360doc 个人图书馆网．软件测试管理需要重视的几个问题.
http://www.360doc.com/content/051231/12/2778_52054.html，2005-12.

[26] 软件测试网．妙用管理工具，巧解测试混乱难题.
http://www.51testing.com/html/43/n-86343.html，2008-7.

[27] 开发频道．程序的静态测试之源程序静态分析．
http://developer.51cto.com/art/200809/88830.htm，2008-09-09.
[28] 周欣，孙家骇，杨芙清．青鸟 C++程序理解工具．计算机工程，2000，26(11).
[29] 站长之家．人工测试全面讲解. http://www.zxbc.cn/html/20071206/30164.html，2007-12-06.
[30] Pawel Leszek．用 Eclipse 平台进行 C/C++开发.
http://www.ibm.com/developerworks/cn/linux/opensource/os-ecc/.
[31] Beth R Tibbitts. Static Analysis in PTP with CDT. IBM Research.
http://www.eclipsecon.org/2008/sub/attachments/Static_Analysis_in_PTP_with_CDT.pdf，
2004-3-9.
[32] wiki. Oink: a Collaboration of C++ Static Analysis Tools. https://github.com/dsw/oink-stack.
[33] 李莹，张琴燕．程序理解．计算机应用研究，2001.
[34] 褚城云．安全编码实践三：C/C++静态代码分析工具 Prefast.
http://blog.csdn.net/chengyun_chu/archive/2008/08/05/2768482.aspx，2009-6-1.
[35] 修行的武者．代码静态分析.
http://blog.donews.com/foxgem/archive/2005/04/23/347444.aspx，2005-4-23.
[36] ITPUB 论坛 龙之丹. PRQA 公司静态代码走查工具 QAC/QAC++.
http://www.itpub.net/viewthread.php?tid=1191555，2009-7-16.
[37] 互动百科网．静态测试.
http://www.hudong.com/wiki/%E9%9D%99%E6%80%81%E6%B5%8B%E8%AF%95#catalog#catalog.
[38] CSDN 网 console_zhao 博客．开源静态代码检测工具 Splint.
http://blog.csdn.net/console_zhao/archive/2006/08/09/1042426.aspx，2006-8-9.
[39] IT 技术博客. splint 安装与配置. http://blog.chinaunix.net/uid-670018-id-2077030.html，2007-02-13.
[40] Turbolinux 知识库. Splint.
http://www.turbolinux.com.cn/turbo/wiki/doku.php?id=splint，2009-4-23.
[41] Scott W Ambler. Introduction to Test Driven Development (TDD).
http://www.agiledata.org/essays/tdd.html，2006-5.
[42] Brian Sun．测试驱动开发全攻略.
http://www.blogjava.net/briansun/archive/2009/02/11/8813.html，2005-7.
[43] 戴建国，郭理，曹传东. JUnit 框架剖析．计算机与数字工程，2008, (8).
[44] 李群．便利的开发工具 CppUnit 快速使用指南.
http://www.ibm.com/developerworks/cn/linux/l-cppunit/，2003-8-3.
[45] CSIP 网．软件测试技术 JUnit 和单元测试入门简介.
http://developer.51cto.com/art/200809/88094.htm，2009-7-08.
[46] 傅毅．单元测试工具在 MFC 编程中的使用问题．舰船电子对抗，2004, 27(5).
[47] 董威．单元测试及测试工具的研究与应用．微型电脑应用，2008, 24(5).
[48] 陈站华．软件单元测试．无线电通信技术，2003, 29(05).
[49] 刘波．基于 CUnit 的自动测试框架．电脑知识与技术，2007, (18).
[50] 梁涛，杜家亮．软件单元测试工具的应用方法.“全国第十届遥感遥测遥控学术研讨会”论文集，2006.
[51] 隋智泉．一种改进的单元测试 JUnit 框架．电脑知识与技术，2007, (12).
[52] IBM 甘志．使用 EclEmma 进行覆盖测试. http://www.uml.org.cn/j2ee/200705143.asp，2007-5.
[53] Eclemma 主页. Java Code Coverage for Eclipse. http://www.eclemma.org/index.html.
[54] 百度空间．代码覆盖率的测试工具——Gcov.
http://hi.baidu.com/haozi2638/blog/item/c01f9a506a13256685352499.html，2008-4-1.

[55] 软件开发与测试实验室. JFCUnit with XML. www.csie.ntut.edu.tw/labsdtl/95-summer/0906-1.pdf.
[56] 谢煜涛等. 用 JFCunit 对 GUI 图形界面进行单元测试. 中国测试，2005, 31(3).
[57] IT 技术博客. JFCUnit 测试 GUI 的一个实例（配置篇、代码篇）.
http://blog.csdn.net/EagleWatch/archive/2005/08/24/464541.aspx，
http://blog.csdn.net/eaglewatch/article/details/465868，2009-7-5.
[58] 郭昕等. JFCUnit 在图形界面单元测试中的应用. 计算机应用，2003, 23.
[59] 林晋等. JFCUnit 在图形用户界面单元测试中的应用与自动化实现. 现代电力，2004, 21(6).
[60] 吴谋硕等. GUI 自动化测试研究. 计算机与现代化，2007, (6).
[61] 致顶网. 如何进行 Java GUI 图形用户界面编程. itpapers.zdnet.com.cn/itpaper/detail/2/13980.shtml，
http://down1.zol.com.cn/wb/33999.doc.
[62] 姜卫，汪厚祥. 图形用户界面的测试自动化. 舰船电子工程，2004, (3).
[63] 刘一帆等. Windows 应用程序界面测试技巧. 计算机应用，2001, 21(8).
[64] 马延平等. Web 测试方法浅析. 电子工程师，2008, 34(11).
[65] 百度文库. Web 页面测试.
http://wenku.baidu.com/link?url=0AwAenvbL1ZSgQGkAxjyISenG7-a-l65hb3v6jn1HPT2Crn2tHFSmsO_1g18K2Az4SFR1jN94Tkh-Y4R2iLYk8C4hg9nDQRXY24vewmLAGa，2007-11-14.
[66] 软件开发网. 使用 HttpUnit 进行集成测试.
http://www.mscto.com/testing/200811219232.html，2008-11.
[67] 测试时代采编. JWebUnit 为 Web 应用程序创建测试用例的办法.
http://www.ltesting.net/html/10/n-163810.html，2009-6-17.
[68] Amit Tuli IBM DW 中国. JWebUnit 框架让测试 Web 应用程序变得轻而易举.
http://fanqiang.chinaunix.net/app/web/2005-06-10/3296.shtml，2005-6-10.
[69] CSDN 博客网. JWebUnit 做 Web 项目自动化测试.
http://blog.csdn.net/plainfield/archive/2007/07/02/1675546.aspx，2007-7-2.
[70] Gerd 主页. Gerd. http://testbit.eu/~timj/historic/gerd.
[71] 李茜等. EASTT：一种嵌入式应用软件测试系统. 计算机工程与科学, 2002, 24(2).
[72] 李茜，凌辉，许晓春等. 一组基于三级度量模型的面向对象度量准则，计算机科学, 2001, (2).
[73] 许蕾等. Web 测试综述. 计算机科学，2003，30(3):100-104.
[74] 兰景英等. Web 系统性能测试研究. 计算机技术与发展，2008, 18(11).
[75] 黄锋等. Web 应用性能测试工具研究与实现. 计算机工程与设计，2008, 29(13).
[76] 杨莲等. Web 应用性能测试研究. 电脑知识与技术，2008, 2(11).
[77] 施卫娟等. 基于 WAST 的 Web 网站压力测试. 电脑知识与技术，2008, 3(5).
[78] 常广炎. 基于 Web 系统的性能测试. 办公自动化杂志，2008, 2.
[79] 王启荣等. 通用 GUI 及其在性能评测开发中的应用. 计算机应用研究，2005, (1).
[80] 高春阳等. 一种基于 JWebUnit 的 Web 应用程序测试方法. 中国科技论文在线，2009-2-24.
[81] 软件测试网. WeBload 工具介绍.
http://www.51testing.com/?uid-105508-action-viewspace-itemid-75192，2008-2.
[82] 百度空间. Web 网站的性能测试工具整理.
http://hi.baidu.com/seowatch/blog/item/53703b467cfdd90e6a63e5c1.html，2008-12-16.
[83] 硅谷动力. 服务器性能测试典型工具介绍.
http://www.enet.com.cn/article/2009/0702/A20090702494179.shtml，2009-7-2.
[84] 邵燕琳. Web 系统性能测试工具的研究. 内蒙古大学硕士学位论文，2007, 6.
[85] 张黄瞩. 认识 p-unit: 一款开源的性能测试工具.
http://www.ibm.com/developerworks/cn/java/j-lo-punit/index.html，2007-5-31.

[86] 聚杰网. p-unit: 性能测试工具. http://softtest.chinaitlab.com/qita/746794_3.html，2008-5-23.
[87] 开发频道. 认识 Web 网站的性能测试工具.
http://developer.51cto.com/art/200811/97318.htm，2009-2-11.
[88] 胡键. 使用 JMeter 完成常用的压力测试.
http://www.ibm.com/developerworks/cn/opensource/os-pressiontest，2006-6-26.
[89] Open 文档. JMeter 基本使用方法.
http://www.open-open.com/doc/view/299b3fd112a84a89ae63f4aee36fef1c，2008-05-27.
[90] 肖菁. 使用 JMeter 进行性能测试. http://www.blogjava.net/jacky/articles/36291.html，2004-2-13.
[91] 软件测试网. 性能测试工具的原理. http://www.51testing.com/html/44/n-70844.html，2007-12-21.
[92] 中国 IT 实验室许高飞. 性能测试工具的介绍.
http://softtest.chinaitlab.com/xn/757444.html，2008-7-24.
[93] 泽众软件. 性能测试工具在测试工作中的重要性.
http://www.spasvo.com/html/ceshi/20080613-144.html，2008-6-13.
[94] CSAI 网. 软件全生命周期工程支撑与量化质量测评管理系统 Panorama++ 简介.
http://se.csai.cn/casepanel/200801021457431582.htm，2008-1-2.
[95] 陈兵. 软件评测完善工程质管体系. 中国质量报, 2008,4.

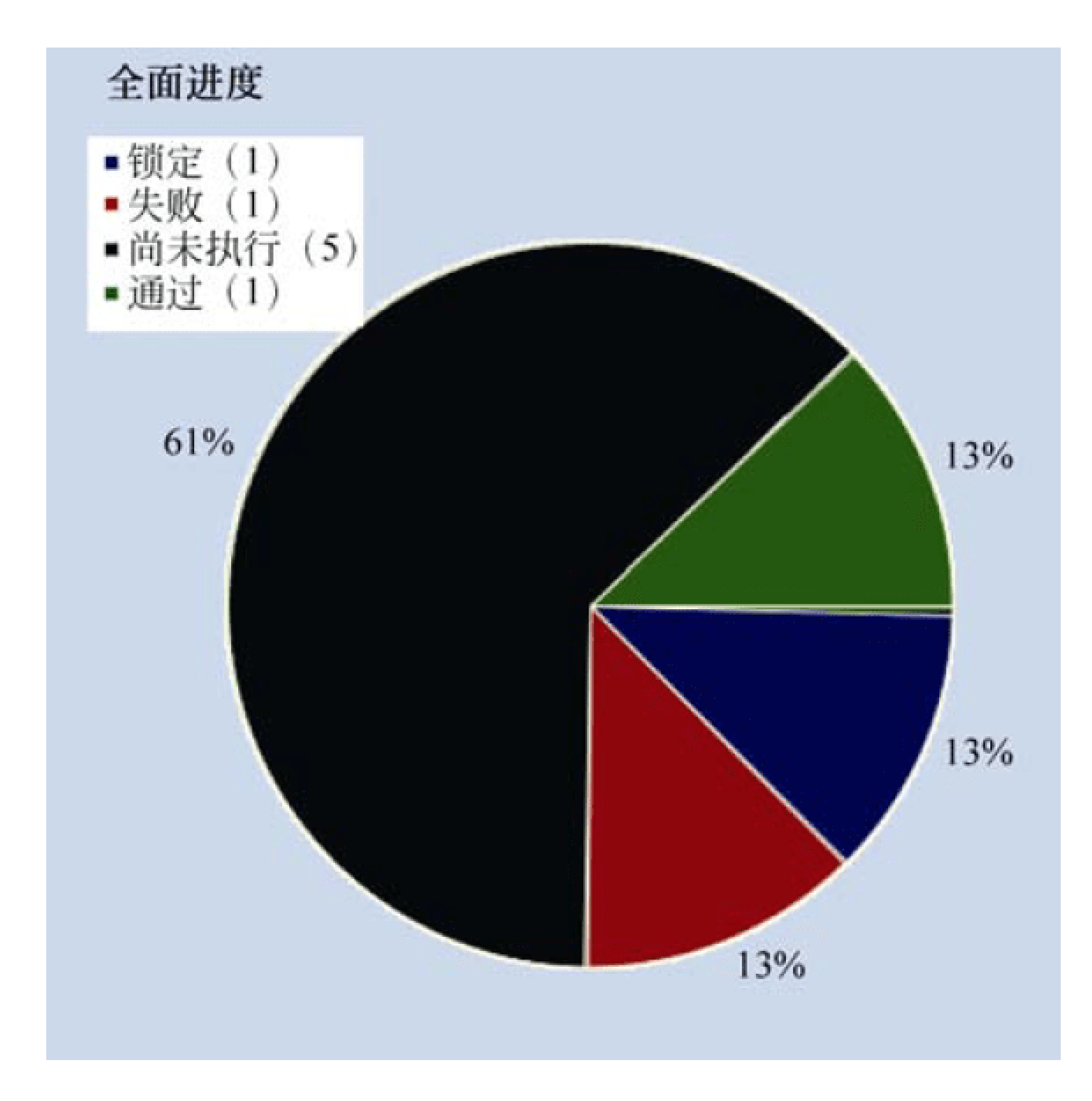


图 2-42　总体测试结果饼图

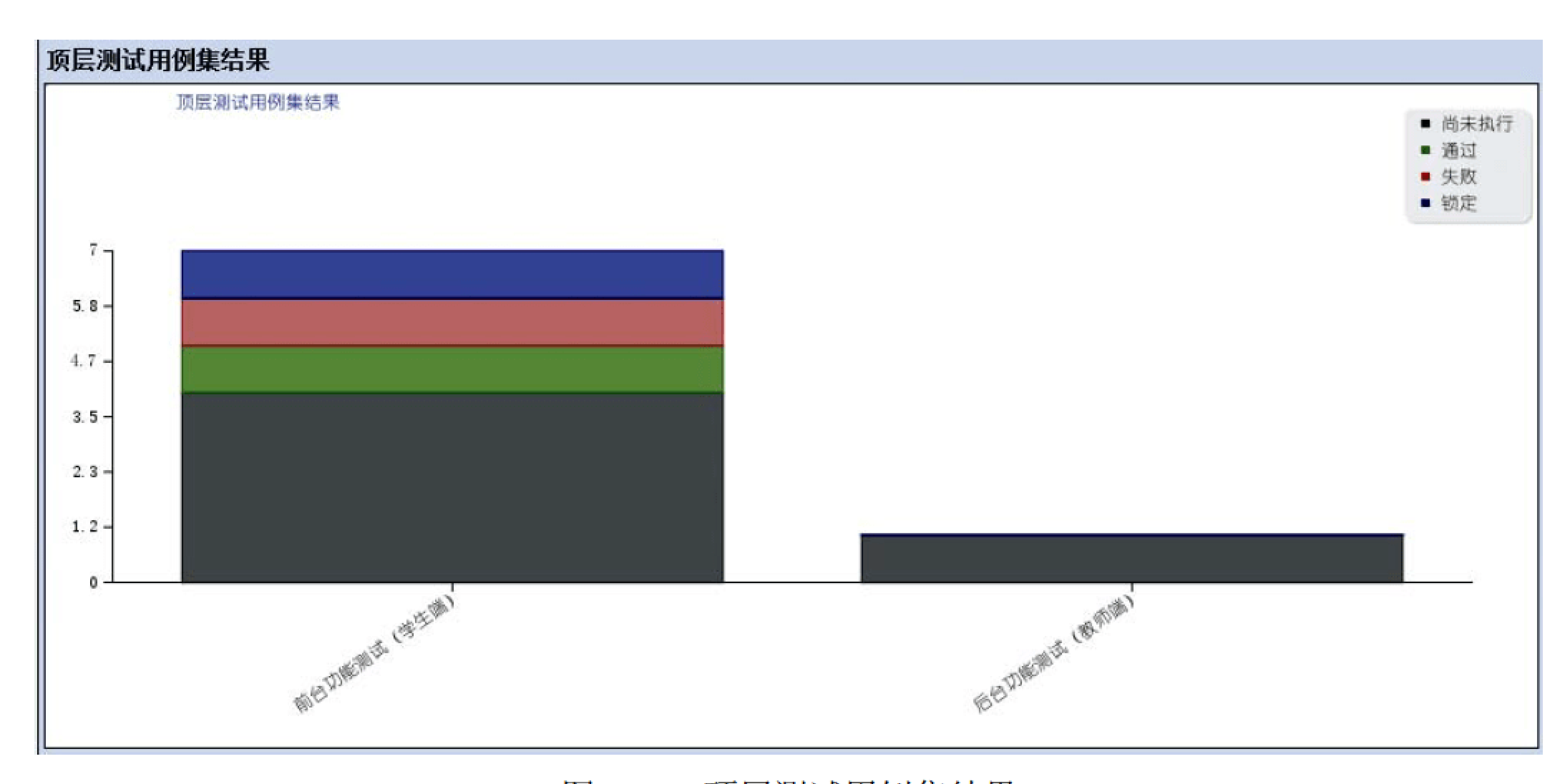


图 2-43　顶层测试用例集结果

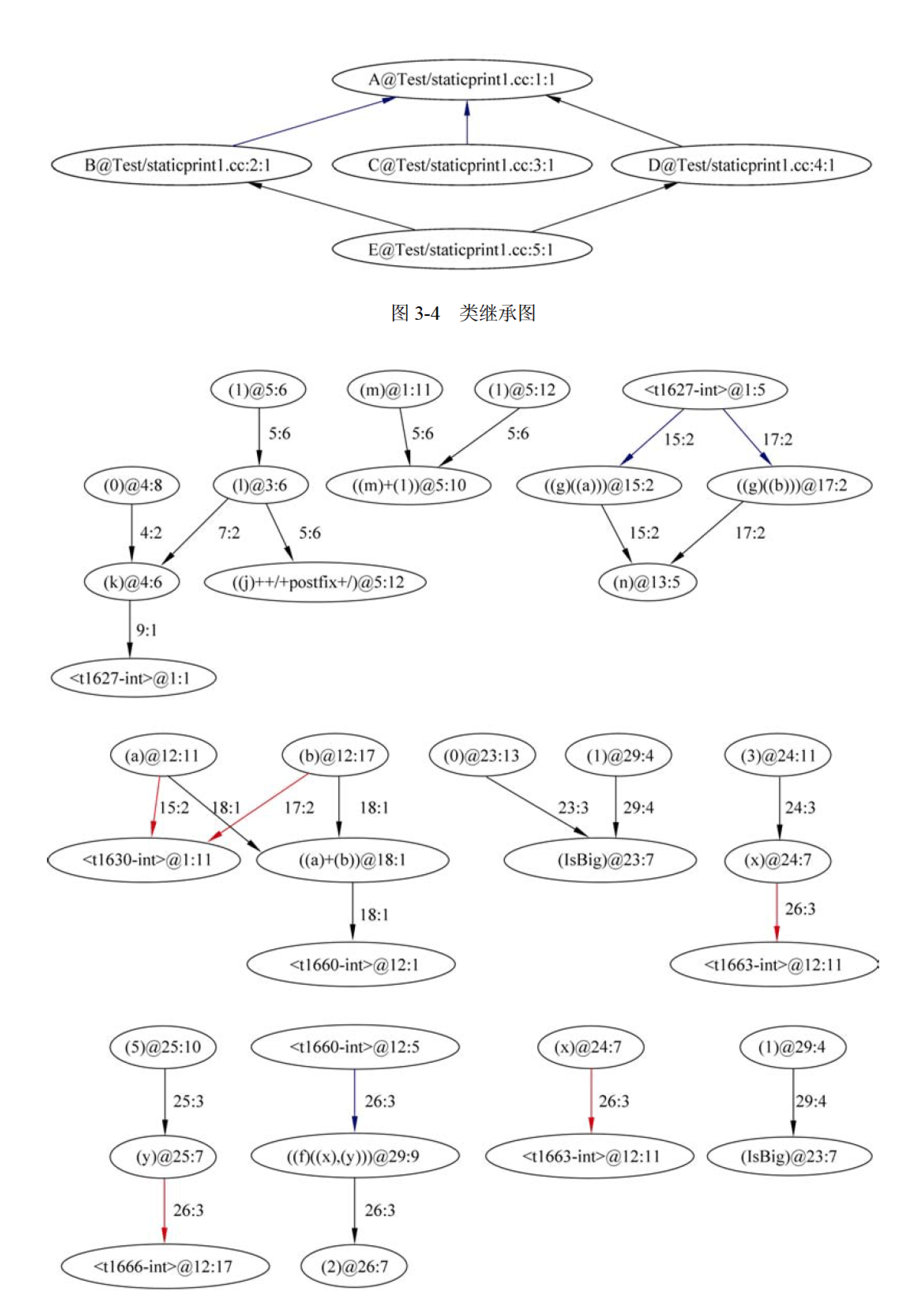


图 3-4　类继承图

图 3-6　程序示例 1 的数据流图

HelloWorld.java

```java
public class HelloWorld {

    public static void main(String[] args) {
        int rand = (int) (Math.random()*100);
        if(rand%2==0){
            System.out.println( "Hello, world! 0");
        }
        else
            System.out.println("Hello, world! 1");

        int result = rand%2==0? rand+rand:rand*rand;
        System.out.println(result);
    }
}
```

图 6-3　覆盖测试结果显示

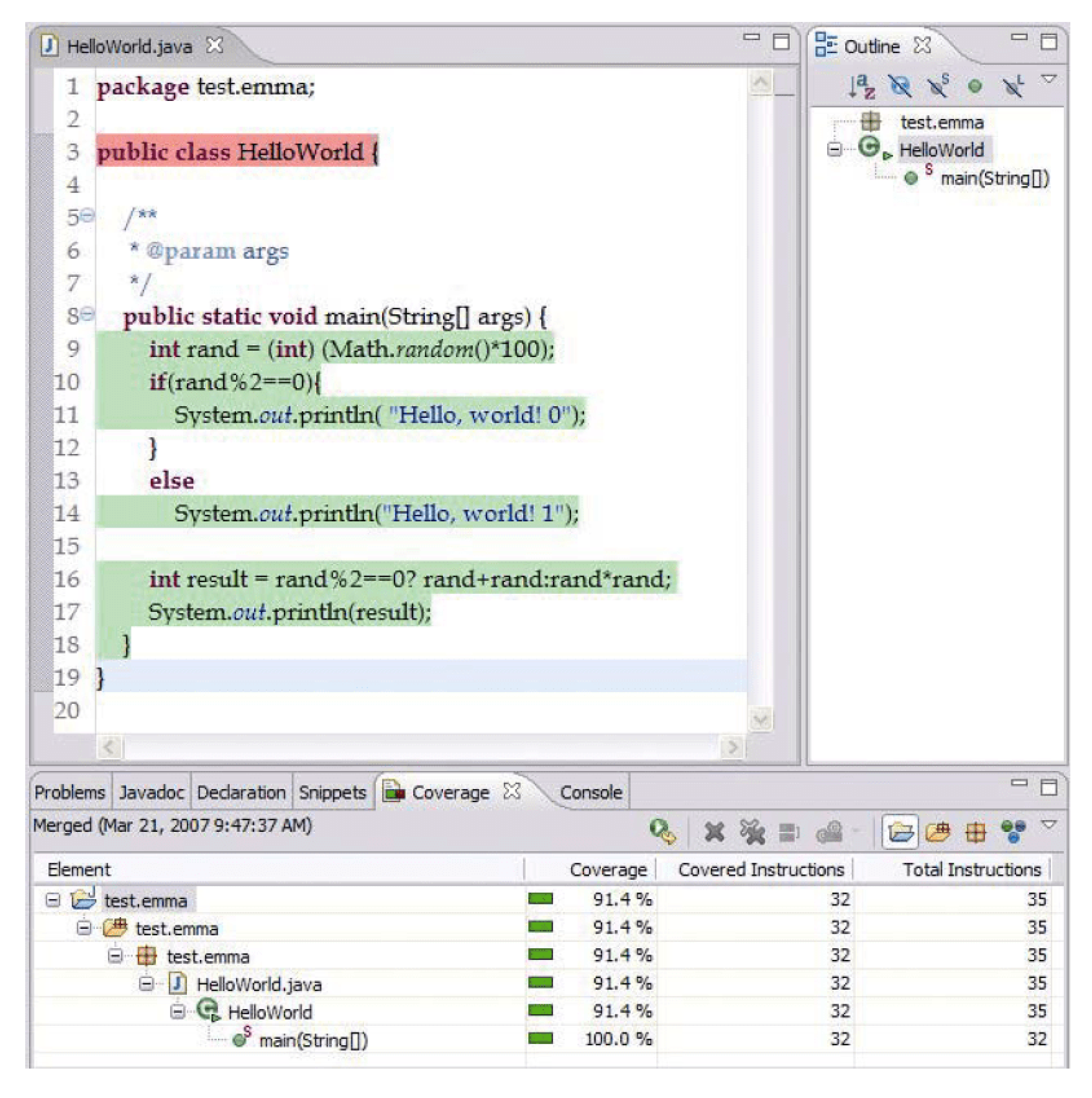


图 6-9　覆盖测试合并结果

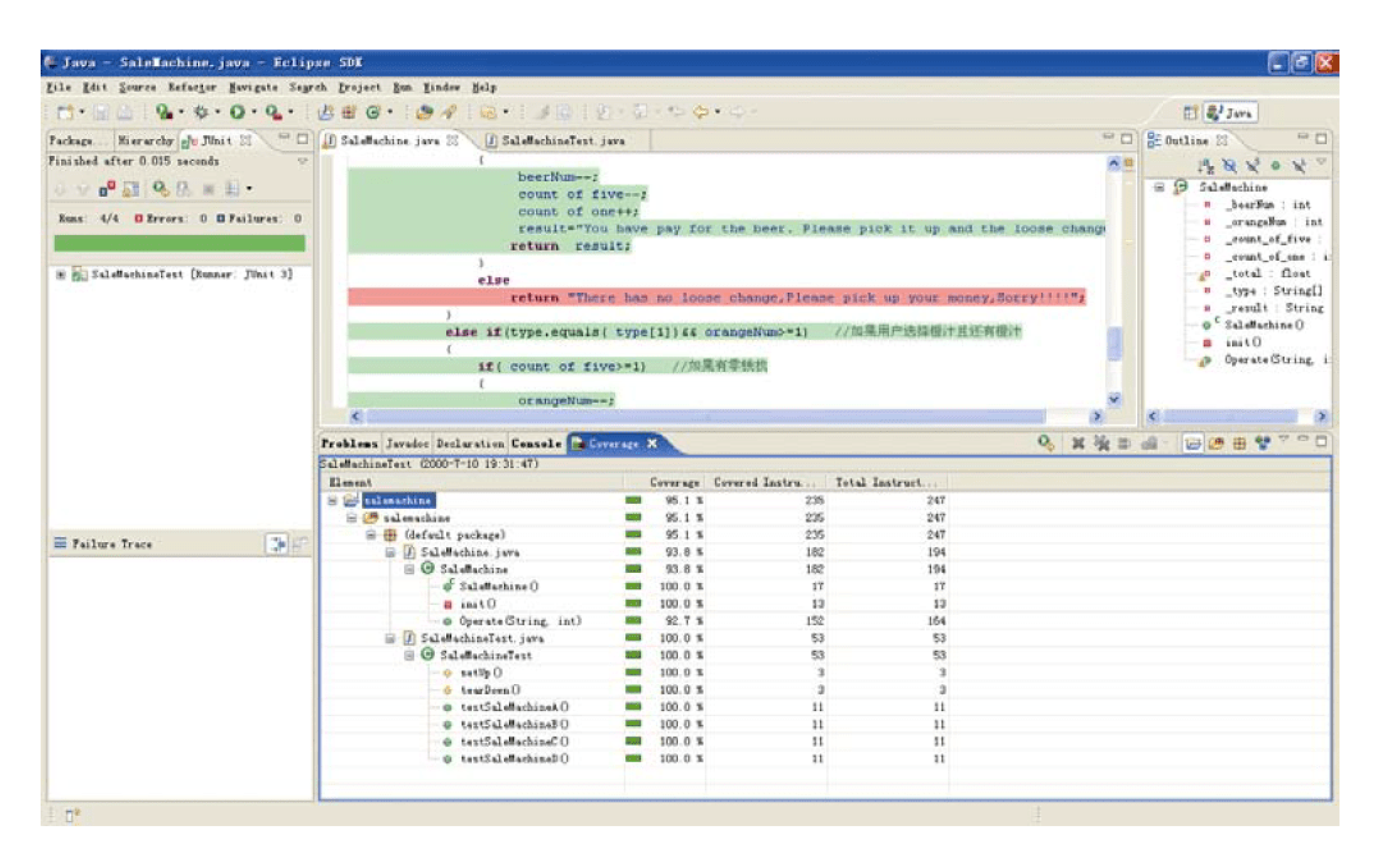


图 6-12　顾客能够买到饮料的覆盖测试结果

SaleMachine.java　SaleMachineTest.java

```
            else
                return "The type message is errno!!!";
        }
        else if(money==1)    //如果用户投入一元钱
        {
            if(type.equals( type[0])&& beerNum>=1)   //如果用户选择啤酒且还有啤酒
            {
                if( count of five>=1)    //如果有零钱找
                {
                    beerNum--;
                    count of five--;
                    count of one++;
                    result="You have pay for the beer. Please pick it
                    return  result;
                }
                else
                    return "There has no loose change,Please pick up your money,Sorry!!!!";
            }
            else if(type.equals( type[1])&& orangeNum>=1)   //如果用户选择橙汁且还有橙汁
            {
                if( count of five>=1)    //如果有零钱找
                {
                    orangeNum--;
                    count of five--;
                    count of one++;
                    result="You have pay for the orange. Please pick it up and the loose cha
                    return  result;
                }
                else
                    return "There has no loose change,Please pick up your money,Sorry!!!!!";
            }
        }
        return "There has some input error!!!!!";
    }
```

还有红的部分需要增加测试用例来使得所有的部分都变绿！！

图 6-14　还需补充的测试用例警告